NONCONFIDENTIAL
No. 14-1135

United States Court of Appeals for the Federal Circuit

CARDSOFT (ASSIGNMENT FOR THE BENEFIT OF CREDITORS), LLC,
Plaintiff-Appellee,

– v. –

VERIFONE, INC., HYPERCOM CORP., VERIFONE SYSTEMS INC.,
Defendants-Appellants.

ON APPEAL FROM THE UNITED STATES DISTRICT COURT FOR THE EASTERN DISTRICT OF TEXAS, CASE NO. 08-CV-00098
HON. ROY S. PAYNE & HON. CHAD EVERINGHAM, M.JJ.

OPENING BRIEF AND ADDENDUM OF DEFENDANTS-APPELLANTS

Robert W. Kantner
Jones Day
2727 North Harwood Street
Dallas, TX 75201
(214) 220-3939

E. Joshua Rosenkranz
Mark S. Davies
Richard A. Bierschbach
Brian D. Ginsberg
Cam T. Phan
Orrick, Herrington & Sutcliffe LLP
51 West 52nd Street
New York, NY 10019
(212) 506-5000
jrosenkranz@orrick.com

Attorneys for Defendants-Appellants

CERTIFICATE OF INTEREST

Counsel for appellants certifies the following:

1. We represent VeriFone, Inc., VeriFone Systems Inc., and Hypercom Corp.

2. VeriFone, Inc., VeriFone Systems Inc., and Hypercom Corp. are the real parties in interest.

3. VeriFone, Inc. is the operating entity of, and is wholly owned by, VeriFone Systems Inc. VeriFone Systems Inc. is a publicly traded company. No publicly traded company owns more than 10 percent of VeriFone Systems Inc. And no publicly traded company other than VeriFone Systems Inc. owns more than 10 percent of VeriFone, Inc.

Hypercom Corp. is a wholly owned subsidiary of VeriFone Systems Inc.

4. The names of all law firms and the partners and associates that appeared for VeriFone, Inc. and VeriFone Systems Inc. in trial court or agency, or are expected to appear in this court are:

ORRICK, HERRINGTON & SUTCLIFFE LLP

E. Joshua Rosenkranz
Mark S. Davies
Richard A. Bierschbach
Brian D. Ginsberg

Cam T. Phan

JONES DAY

Robert W. Kantner

KOLISCH HARTWELL PC

Brantley C. Shumaker
David C. Bourgeau
Owen W. Dukelow
Peter E. Heuser
Shawn J. Kolitch

POTTER MINTON P.C.

E. Glenn Thames

The names of all law firms and the partners and associates that appeared for Hypercom Corp. in trial court or agency, or are expected to appear in this court are:

ORRICK, HERRINGTON & SUTCLIFFE LLP

E. Joshua Rosenkranz
Mark S. Davies
Richard A. Bierschbach
Brian D. Ginsberg
Cam T. Phan

JONES DAY

Robert W. Kantner

PARKER, BUNT & AINSWORTH PC

Robert C. Bunt

DLA PIPER US LLP

Aaron G. Fountain
William G. Goldman

GILLAM & SMITH, LLP

Melissa Richards Smith

FEINBERG DAY ALBERTI & THOMPSON LLP

Sal Lim

Date: February 18, 2014

Respectfully submitted,

/s/ E. Joshua Rosenkranz

E. Joshua Rosenkranz
Orrick, Herrington & Sutcliffe LLP
51 West 52nd Street
New York, NY 10019
(212) 506-5000
jrosenkranz@orrick.com

Attorney for Defendants-Appellants

TABLE OF CONTENTS

Page

Material has been deleted from pages 10, 11, 18, 42, and 43 of the nonconfidential Opening Brief of Defendants-Appellants VeriFone, Inc., VeriFone Systems Inc. and Hypercom Corp. This material is deemed confidential information pursuant to the Protective Order entered October 1, 2009. The material omitted from these pages contains confidential business information.

ADDENDUM

TABLE OF AUTHORITIES

Page(s)

Federal Cases

Case: 14-1135 Document: 28 Page: 9 Filed: 02/18/2014

Federal Statutes

Other Authorities

Case: 14-1135 Document: 31 Page: 12 Filed: 02/18/2014

STATEMENT OF RELATED CASES

There are no related cases pending in this Court. The following related cases are pending in the U.S. District Court for the Eastern District of Texas: *CardSoft (Assignment for the Benefit of Creditors), LLC, v. Hypercom Corporation and VeriFone, Inc.*, No. 2:13-cv-00941-RSP; CardSoft *(Assignment for the Benefit of Creditors), LLC v. First Data Corp.*, No. 2:13-cv-00290-JRG-RSP; and *CardSoft (Assignment for the Benefit of Creditors), LLC v. The Gores Group, LLC*, No. 2:12-cv-00325-JRG-RSP.

INTRODUCTION

CardSoft, a software company, patented and attempted to market a virtual machine product it hoped would change the payment-terminal industry. That did not happen. The industry—including the three leading payment-terminal manufacturers, VeriFone, Inc. ("VeriFone"), Hypercom Corporation ("Hypercom"), and Ingenico Corporation ("Ingenico")—rejected the software completely. They concluded that CardSoft's product was at odds with their large investments in customized terminals, with the real needs of the merchants who put terminals in stores and other businesses, and with the security concerns of customers who enter their sensitive personal and financial information into those terminals to pay for goods and services. Hemorrhaging cash, and with no market for its product, CardSoft effectively closed up shop and assigned its assets to a group of creditors. In a last-ditch attempt to find some money, those creditors sued the manufacturers for infringing the very patent that described the virtual machine the manufacturers had rejected. And, through a badly flawed claim construction, CardSoft's desperate tactic succeeded.

The claim construction error was fundamental and requires reversal. CardSoft's "virtual machine" was software that could run applications that did not depend on any specific underlying operating system or hardware. The magistrate judge who construed the claim, however, concluded that the virtual machine need not run such applications. The magistrate did so because he repeatedly confused the *virtual machine* itself with the *applications* the virtual machine runs. Based on the mistaken claim construction, the jury was permitted to impose a multi-million dollar verdict and the magistrate granted CardSoft's request for ongoing royalties.

This Court reviews claim construction decisions *de novo* because it has a specialized role in the patent system. Proper claim constructions are critical to a system that promotes innovation rather than provides windfalls to litigious plaintiffs. Reversal is warranted.

JURISDICTIONAL STATEMENT

The magistrate judge had jurisdiction over this case pursuant to 28 U.S.C. §§ 1331 and 1338(a). He entered final judgment on October 30, 2013. A100. The notice of appeal was timely filed on November 26,

2013. A19,136-37. This Court has jurisdiction to decide this appeal under 28 U.S.C. § 1295(a)(1).

STATEMENT OF THE ISSUES

The questions presented are:

1. Must the claimed "virtual machine" run applications that do not depend on any specific operating system or hardware?

2. Did CardSoft prove that the accused terminals "emulate a hypothetical computer" when the only affirmative evidence is a single conclusory statement?

STATEMENT OF THE CASE

VeriFone And Hypercom Thrive By Building A Wide Range Of Payment Terminals That Cannot Share Applications

Defendants-Appellants VeriFone and Hypercom manufacture payment terminals.[1] We have all used them. They are the machines that let us conveniently swipe or scan credit and debit cards. *See* A16,989-90. They come in many shapes and sizes, such as:

[1] VeriFone and Hypercom were competitors and co-defendants when this case was initiated. In August 2011, Hypercom became a wholly-owned subsidiary of VeriFone. For clarity's sake, this brief refers to VeriFone and Hypercom separately.

VeriFone and Hypercom got into the payment-terminal business in the early 1980s. A17,492, 17,511. Through innovation and competition, they have become industry leaders; along with Ingenico, another manufacturer, they account for about two-thirds of the payment-terminal market. A13,056, 17,528. As we explain further below, the industry leaders have thrived by providing payment terminals that are customized to fit particular customer needs. Largely because of their efforts, various types of payment terminals are now everywhere: big-box retailers, grocery stores, restaurants, hotels, movie theaters, gas stations, taxicabs, and a host of other businesses. *See, e.g.*, A17,505, 17,523-24, 17,604, 17,608, 17,629-30.

Payment Terminal Technology. Technologically speaking, each payment terminal is a computer. A17,134. Like many computers, each

terminal has its own hardware, operating system, and application programs. A115-16, col. 2:67-3:2. Together, they determine exactly what the terminal can do. *Id.*; A116, col. 3:4-5.

The hardware consists of the terminal's physical components. The most significant piece is the processor—the terminal's "brain"—housed inside the terminal's main unit. *See* A103, fig. 1; A115, col. 2:64-65. The hardware also includes peripherals, e.g., the card-reader, keypad, stylus pen and signature pad, and monitor depicted in the images above. A103, fig. 1; A115, col. 2:65-67; A119, col. 9:9-10.

Each terminal also has application software. An application is a set of "instructions" that provide directions for the terminal's operation. A119, col. 10:40-47. The application software in payment terminals consists of programs dedicated to specific tasks like credit card transactions, debit card transactions, customer loyalty programs, and the like. A115, col. 2:26-29; A116, col. 3:29-32; A17,074.

In payment terminals, just as in other computers, the hardware and software interact. They do this through the operating system. A119, col. 9:37-45. The operating system manages the hardware resources and ensures that all the applications function properly

together. A13,337, 17,627-28. When a customer uses a credit card to purchase laundry detergent from CVS, for example, one application might direct the hardware to "Print a receipt showing that the customer paid $7.99 for laundry detergent." *See* A119, col. 9:35-36. At the same time, a separate customer loyalty application could instruct the hardware to "Print this week's customer loyalty coupons." *See* A17,661. The operating system triages these instructions and passes them through to the appropriate hardware components at the appropriate time for the hardware to execute the instructions. A13,337, 17,627-28.

A Terminal Can Only Run Applications Written For Its Specific Underlying Operating System And Hardware Configuration. The hardware and software of conventional payment terminals lead to some fundamental requirements when it comes to construction of the terminals. First, the application programs, operating system, and hardware all must be able to communicate with each other. *See* A16,983, 17,074. Otherwise, the operating system will not be able to successfully convey the application's instructions to the hardware and actually get things done. *See* A17,626-27. *See generally* James E. Smith & Ravi Nair, *Virtual Machines: Versatile Platforms for*

Systems and Processes 2-3, 6-10 (2005). Second, even if they can communicate in theory, the hardware and operating system must actually have the capability to carry out the specific tasks that an application asks it to do—for example, "Print a receipt." Otherwise, the application will not work. *See* A17,628-29.

This means that applications are written "specifically for each type of device" and are "not portable between devices having different hardware or operating system architectures." A119, col. 9:48-51. Instead, "you need different instructions on different equipment." A4748. An application that is written to work on one operating system will not work on a terminal running a different operating system, just as a program written for a Mac operating system will not run on a Windows one. A17,507. Similarly, even if an application can run on a given operating system, that operating system itself might not work with all hardware, just as the Mac operating system will not run on a Dell computer. A17,503-07.

As a result, whenever a programmer wants to write an application for a particular type of terminal, the programmer has to go through several steps:

(1) Write the instructions that make up the application using a terminal-specific "Development Tool Kit", which contains "startup code, … libraries[2] and related tools required for the application[] to run on" that specific terminal. A14,150, 14,158. The instructions will be in what is called "source code," which just means a human-readable programming language that takes the form of text that humans can understand. A17,130-31. For VeriFone, the "[a]pplications for Verix and Verix V terminals are written in C or C++ programming languages." A13,337, 14,166.

(2) Convert the human-readable instructions into the computer's own "native code." In order for a machine to understand instructions that are written by a human, the instructions need to be compiled into the native code of the machine, i.e., in code that the machine can read. A17,045. The programmer does so by using what is called a "compiler." "Compiling" is the process of "tak[ing] programme instructions and translat[ing] them into the right form for the microprocessor." A4748. Because of the variety of forms of microprocessors, each terminal-

[2] Libraries "are collections of software code that perform common software functions." Ron White, *How Computers Work* 102 (8th ed. 2006).

specific hardware-operating system configuration requires its own special compiler.

(3) Once the instructions have been written and converted to native code, the programmer must place them on the terminal. The compiling process produces an application file containing the native-code instructions, which the programmer downloads to the terminal. A4586, 13,337, 14,166.

(4) The programmer has to repeat these steps for every different type of terminal, rewriting and recompiling and downloading the same application so that it will work on the terminal's particular operating system-hardware setup.

One thus cannot simply load a VeriFone application written for one terminal configuration onto another VeriFone terminal with a different configuration and expect it to work anymore than one can run Microsoft Word for Mac on a PC. A17,507. In each case, a programmer has to once again write, compile, and download the application to work with the precise operating system and hardware configuration at issue. A17,521.

Confidential Material Omitted

This is no small task. VeriFone, for instance, uses six operating systems—Verix, Verix V, Verix eVo, Linux, TXO, and NOS—each of which it spent thousands of hours developing, or in the case of the "commercially available" Linux operating system, "heavily modify[ing]." A17,503-07. Each operating system requires a particular processor. For example, Verix uses the 68000 and Linux uses the ARM 9. *Id.* The Verix operating system will not run on the ARM 9 and, by the same token, an application written for the Verix eVo operating system will not run on the Verix V operating system. *Id.*; A17,520-22. Hypercom also has its own array of terminals running proprietary operating systems that require particular processors. A8115.

These charts show both manufacturers' complete range of operating systems and corresponding processors, along with some of the different types of terminals for which they are configured: [redacted]

[redacted] *See* A4649. [redacted]

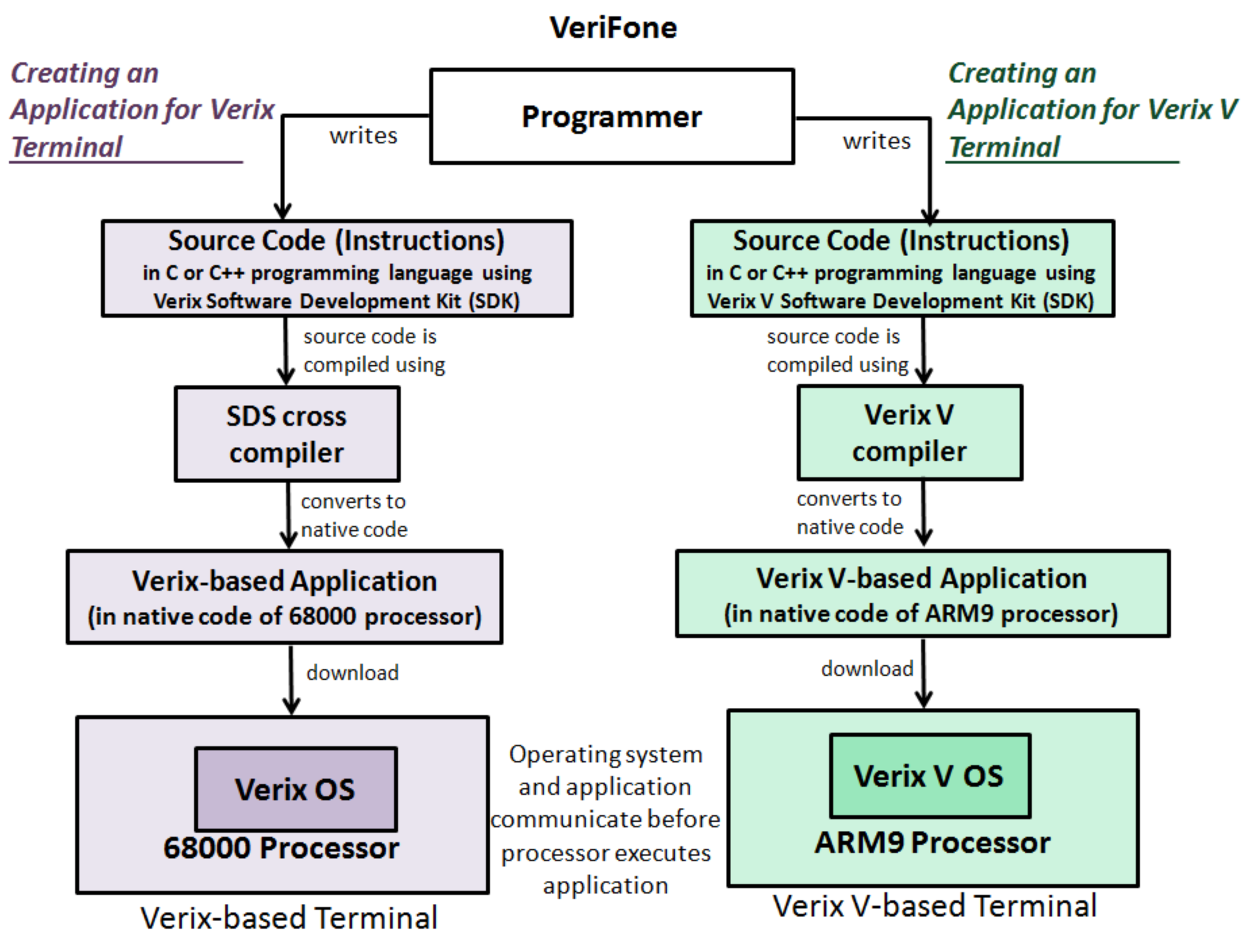
[redacted]

[redacted] *See* A8115.

As mentioned above, programmers use a variety of different files and other "tool kits" to aid them in the process of developing and writing applications to work on a specific type of terminal. A14,150. To create applications that will work with the Verix operating system and 68000 processor, for example, a programmer needs to use the "Verix SDK" (Software Development Kit) made just for Verix terminals and the "SDS cross compiler" to compile the instructions into the 68000's native code. A14,151, 14,158, 14,166. To write the same application for the Verix V-based operating system and ARM9 processor, a programmer needs the "Verix V SDK" and "the Verix V Compiler." A14,158, 14,167. Thus, for example, for VeriFone's popular SoftPay application to run on all VeriFone terminals, VeriFone has to "essentially change the source code and compile it to [the] specific

platform" for each different type of terminal. A17,523. Similarly, to create an application for Hypercom's "T7-family" terminals utilizing Hypercom's proprietary OS and the Zilog Z80 processor, A14,473, a programmer needs the "Hypercom Software Development Kit," A14,457, and "Z80 cross-compiler," A14,468.

Schematically, the conventional steps necessary to write and execute the same application on multiple types of VeriFone terminals with different hardware and operating system configurations—say, one using the Verix operating system and the 68000 processor, and one using the Verix V operating system and the ARM9 processor—look like this:



See A14,158-59, 14,166-68.

The Business Model. VeriFone, Hypercom, and Ingenico have used the above approach to terminal construction since they entered the business over three decades ago. A17,511. They have done so with great success: Indeed, it is the very foundation of their business model. When a merchant like CVS contacts one of them to order a comprehensive point-of-sale solution, the manufacturer and CVS build it together. First, they choose the underlying terminal. Next, they

decide on a suite of applications, and the manufacturer develops and configures them to run on the underlying terminal. A17,523-24, 17,561-63. And then the manufacturer secures the terminal by locking it to prevent any sort of software modification. A14,389. When any part of the terminal has to be updated, including altering just a single byte of a single application, the manufacturer, not the merchant, has to do it. A17,524.

VeriFone, Hypercom, and Ingenico are continuously researching and developing innovative terminal architectures. Merchants in different industries have different point-of-sale payment needs, and the manufacturers make a huge range of terminals to accommodate them. A17,607-08, 17,631, 17,661-62, 17,664. For taxis, the manufacturers make no-frills terminals that prioritize efficiency and simplicity. A17,372. For large retailers, they make sturdier machines with more peripherals and more sophisticated applications to facilitate what tends to be a more complex customer checkout process. These terminals often have color- and video-enabled touchscreen monitors, high-resolution printers, and other premium features. *See, e.g.*, A17,505-06. Because the terminals are "design[ed] and programm[ed]" differently, A17,031,

the manufacturers are required to "create different software," including "customized app[lication]s" and operating systems for the different types of terminals. A17,523-24; *see also* A16,991, 17,034, 17,605.

This "custom approach" to terminal construction, A17,031, makes it highly unlikely that any two terminal types can execute the same application without modification. So long as there is any variation in the operating system or hardware configuration between terminals, an application written for one configuration will always need some reprogramming to run properly on another. A17,523-24. Thus, at a minimum, for an application programmed to run on one particular terminal configuration to work on another, a programmer would have to "recompil[e] the source code to target a different device." A83, 17,504, 17,521, 17,823-25.

CardSoft Develops A "Virtual Machine" Designed To Run Applications That Do Not Depend On Any Specific Underlying Operating System Or Hardware

In 1996, Plaintiff-Appellee CardSoft, a software company, began working on a product that it thought would improve the terminal-manufacturing business. A17,018. CardSoft's founder and CEO, Ian Ogilvy, did not like the prevailing custom-tailored approach to terminal

construction. He thought that merchants who needed to update or change applications on their terminals should be able to seek out third-party solutions. *See* A17,035. He also thought that the approach created a lot of "wasted work." A17,007. He believed that so many merchants needed the same applications that programmers were consumed with "rewrit[ing] [them] again and again" to work on different terminal configurations. A17,006.

While Ogilvy was at the drawing board, Sun Microsystems, Inc., a computer software company, was introducing the now-ubiquitous Java Virtual Machine. A17,020. Once installed, the Java Virtual Machine allows a computer to run any applications written in the Java language regardless of that computer's underlying operating system or hardware configuration. It does so by processing instructions written in "a sort of Esperanto that is not machine specific but that can quickly be interpreted by any computer" regardless of its particular hardware/operating system arrangement. David Bank, *The Java Saga*, Wired (Dec. 1995), *available at* http://www.wired.com/wired/archive/3.12/java.saga.html. In other words, the Java Virtual Machine does not care what kind of machine you have. *Id.* Or, as Java's slogan

put it, the Java Virtual Machine heralded the "write once, run anywhere" approach to application development. A13,271.

In an effort to bring "virtual machines" to the payment industry, Ogilvy wrote "CardScript." A4747, 4753. "CardScript is an advanced programming tool" that programmers use to develop applications, or "instructions," for the CardScript virtual machine. A4753-54, 17,003. Unlike conventional applications such as those used on VeriFone's and Hypercom's devices, however, CardScript applications "would not be compiled into the native code of the [terminal's] processor." A17,047. Instead, like Java, CardScript is a language that allows programmers to write applications that do not depend on any specific terminal—in computer science lingo, "a script base[d] cross platform language"—which "means that a program created in CardScript can be used on any payment terminal." A4754.

"To run CardScript applications, each terminal requires a CardScript Driver." A4761. The CardScript Driver is the "CardSoft virtual machine," A4759, i.e., the software that "performs the 'Magic' running the same application program on terminals with different hardware architectures and even different microprocessors," A4761. As

Confidential Material Omitted

long as the CardScript virtual machine is installed on a terminal, any CardScript application can work on that terminal. Thus, one of the "benefits of using CardScript" is "[h]ardware independence" such that "the same application can run on a variety of terminals." A4754. There is no need to convert or "compile" the application into a format that a particular type of terminal can read. A17,047.

Here is how it worked: The application would give instructions to the CardScript virtual machine, say, "Print a receipt showing that the customer paid $7.99 for laundry detergent." The CardScript virtual machine would then carry out the request by translating the "print" instructions from the generic application into something the underlying terminal hardware and software could use. A119, col. 9:67-10:2. This translation, from [redacted] A4545, meant that an application only needs to communicate with the CardScript virtual machine via its virtual processors, A116, col. 3:40-47, thus allowing the application "to operate independent of the [underlying] processor," *id.*, col. 3:34-36; A17,041-43.

Of course, this does not mean that using CardScript involves no work at all. In order to run on a terminal, the CardScript virtual machine —itself an "application," A4762—must still be compiled into the native code of the particular terminal's processor, A17,045, and downloaded to the terminal, A4762. And, using CardScript's own development tool kit, a programmer still has to write and then compile a given application's human-readable source code into the generic code "readable by the [CardScript] virtual machine" before it will work on any CardScript-enabled terminal. A121, col. 14:38-39. But that is *all* the programmer has to do, no matter how many different types of terminals the application is to run on or their different underlying configurations. A17,010-11.

The CardScript virtual machine would, had anyone wanted it, essentially have functioned as a universal adapter for payment terminals that rendered the underlying architecture irrelevant for application programmers. So long as terminal-appropriate versions of CardScript were running on any two, ten, or even one hundred different types of payment terminals, no matter how different their underlying hardware and operating systems, functionally, they each would

simulate the same "CardScript terminal" and would therefore run any application written for the CardScript virtual machine. A programmer who wrote a *single* CardScript application could thus load it onto any of those one hundred CardScript-enabled terminals and have it run properly without ever touching it again.

CardSoft Applies For The '945 Patent

On March 16, 1998, CardSoft filed the patent application that ultimately became U.S. Patent No. 6,934,945 ("the '945 patent"). A102.

Consistent with our earlier description of conventional payment terminals, the patent begins with some background about typical payment terminals. "A payment terminal device usually comprises hardware [and] an operating system … and is ready to accept an application *for that arrangement*." A115-16, col. 2:67-3:2 (emphasis added). The patent observes that the main "operation" of a payment terminal—the processing of payment transactions, such as displaying the purchase amount, prompting a customer to enter their PIN, and sending the information to the bank for authorization, among other tasks—is "handled by the application program." A116, col. 3:14-31. As a result, to "change operation of the machine, the application must be

Case: 14-1135 Document: 28 Page: 33 Filed: 02/18/2014

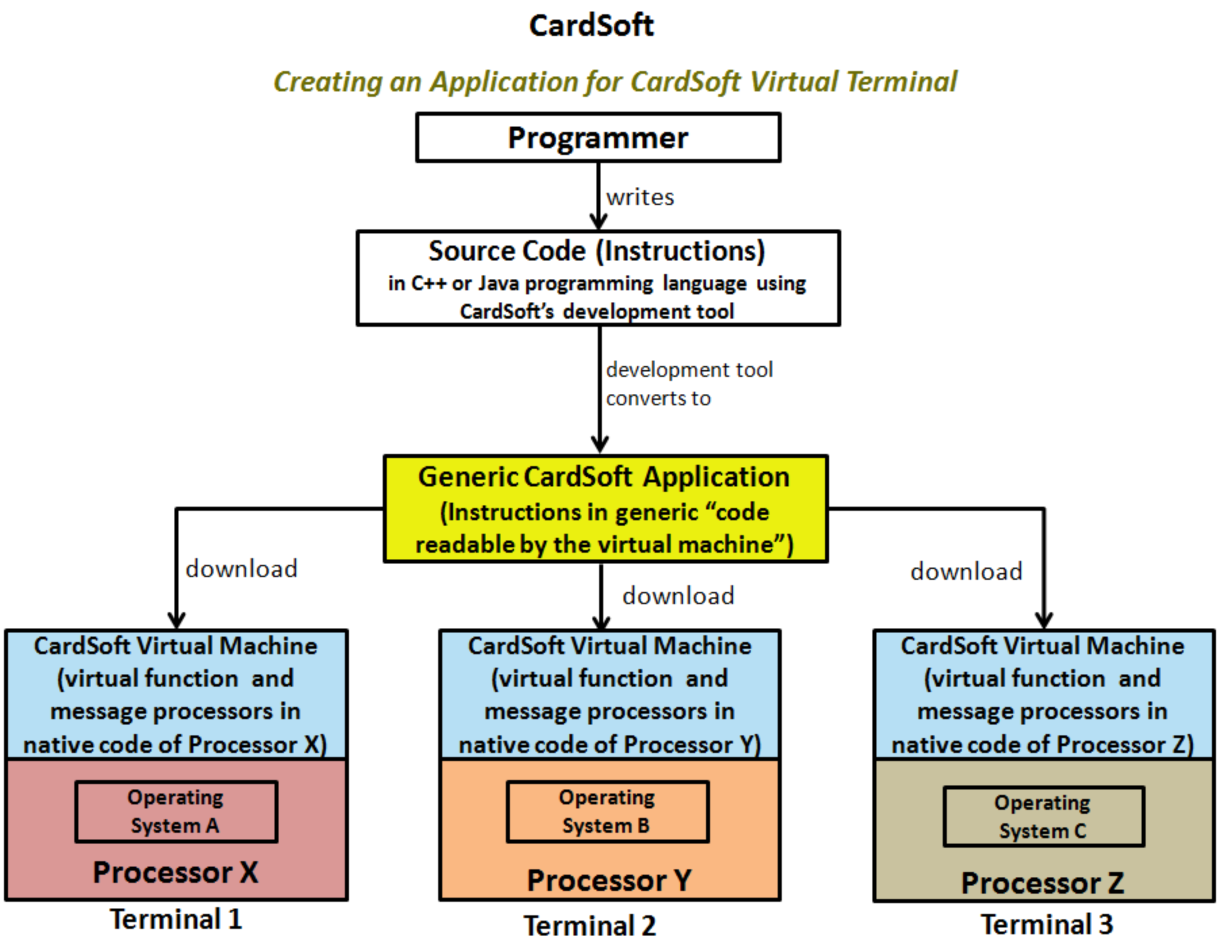
changed." *Id.*, col. 3:31-32. But doing so is not easy. Because "programs are not portable between devices having different hardware or operating system architectures … it is necessary to write a program specifically for each type of device," A119, col. 9:45-51, "even where the same operational alterations may be required," A116, col. 3:5-12. This re-writing, the patent explains, is a significant problem. Since there are "many different brands of payment terminal, utilising many different software and hardware arrangements," A115, col. 2:4-6, it is "time consuming [and] costly" to write "a new application" for "each different type of device," A116, col. 3:5-9. In short, when it comes to applications, one size almost never fits all.

The patent proposes using a CardScript-style virtual machine to solve that problem by ensuring that one size *always* fits all. Like CardScript, the proposed virtual machine has at least two components. First is the virtual machine itself, which includes the "virtual machine processors." A119, col. 9:64-65. As with the "CardScript Driver," A4753-71, the virtual machine processors must be "implemented in native code of the payment terminal," such that each virtual machine needs to be modified for the specific terminal. A119, col. 10:18-29.

Once a virtual machine is modified to work on that payment terminal, any "programs [written] for the virtual computer" can run on that payment terminal. A116, col. 3:44-45. Second is the application that runs on the virtual machine, which includes the protocol and message instructions. A119, col. 10:40-42. The patent describes a development tool for developing application instructions and converting them to the generic code that the virtual machine accepts. A117, col. 6:36-55; A121, col. 14:31, 49-55; A123, col. 17:39-41. Instead of the application being in the native code of the underlying processor, the application is converted to the generic "code readable by the virtual machine." A121, col. 14:37-39. Once in the generic code, the instructions can be downloaded onto and processed by any terminal running the claimed virtual machine. A117, col. 6:47-49; A121, col. 14:37-39. "This creates a complete portable environment for program operations," A116, col. 3:45-46, because the application is written for the virtual machine, not for any particular terminal.

Schematically, the steps necessary to write and execute the same application on multiple types of CardSoft-enabled terminals, each

having a different underlying operating system and hardware configuration, look like this:



See A119, col. 10:18-20; A120, col. 11:11-12; A121, col. 14:31-32, 38-40.

The specification lays out in great detail how the purpose of the claimed device is to ensure that all terminals running it, regardless of their underlying architecture, can process any application instructions written in a single universal language. The claimed virtual machine, it repeatedly explains, "can be implemented on any hardware, BIOS/OS

arrangement,”[3] A117, col. 5:2-3, and all terminals running it “emulate the same hypothetical [terminal],” A116, col. 3:42-43. They all “will thus be capable of executing programs [written] for the virtual computer,” i.e., programs written in a generic language that the virtualization software understands. *Id.*, col. 3:44-45. Thus, “[i]mplementation of such a virtual machine on payment terminal devices of different brands enables operation of the payment terminal devices or brands to be altered merely by altering application commands generic to all brands,” because “[e]ach brand is seen by the application as the same virtual machine.” A117, col. 5:4-9. “Different arrangements of hardware can therefore be controlled by the same application software.” A119, col. 10:5-6.

As relevant here, the patent claims:

> **1.** A communication device which is arranged to process messages for communications, comprising a **virtual machine** means which includes

---

[3] “If the microprocessor is your PC’s brains, [BIOS] is the heart …. The BIOS (Basic Input/Output System) knows the details of how your PC was put together and serves as an intermediary between the operating software running your computer and the various hardware components.” White, *supra* n.2.

a **virtual function processor** and **function processor instructions** for controlling operation of the device, and

message in[str]uction[4] **means** including a set of descriptions of message data;

a virtual message processor, which is arranged to be called by the function processor and which is arranged to carry out the message handling tasks of assembling the messages, disassembling messages and comparing the messages **under the direction of the message instruction means** that is arranged to provide directions for operation of the virtual message processor, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task,

wherein the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems.

A139, col. 50:48-67 (emphasis added).

The '945 Patent Issues Years Later

It took more than six years for CardSoft to secure a patent on its virtual machine because it ran into problems trying to distinguish its product from the Java Virtual Machine that preceded it. A17,053. The patent examiner repeatedly rejected the '945 patent's claims as anticipated by U.S. Patent No. 5,935,249, issued to Stern et al. ("Stern" or "Stern '249"). The examiner consistently found that, like Stern, the

---

[4] The patent refers to "message induction means," but the parties agreed that the correct term is "message instruction means," and the word is rendered "instruction" for the remainder of this brief.

’945 patent disclosed a conventional virtual machine and that CardSoft “failed to clearly disclose the novelty of the invention.” A18,857; *see generally*, A18,839-46, 18,854-60.

CardSoft prevailed only by emphasizing certain features of its claimed virtual machine. Like the virtual machine of the ’945 patent, CardSoft noted, “[d]ifferent versions of the Java Virtual Machine” taught in Stern could be “produced to interface with different underlying processors and operating systems.” A18,907. The “significant[] differen[ce],” CardSoft maintained, is that the Java Virtual Machine did not teach a “dedicated” virtual message processor in the same way that the ’945 patent did. *Id.* The examiner was not convinced. *See* A18,949-55. So CardSoft emphasized that its device also includes an application that manipulates the customer data. A1153. In 2005, after having rejected CardSoft’s application four times, the examiner finally allowed its claims. *See* A18,973.

CardSoft later obtained U.S. Patent No. 7,302,683 on the same virtualization software as applied to devices using smartcards.[5]

[5] The district court and the parties referred exclusively to the ’945 patent when discussing both patents. For ease of reference, this brief does the same.

Case: 14-1135 Document: 28 Page: 39 Filed: 02/18/2014

CardSoft's Product Fails In The Market

While CardSoft's application for the '945 patent was pending, CardSoft had been trying to market its product. Ogilvy knew that changing the industry's successful business practices would be tough. *See* A17,031-32. Yet, in 1999, he decided to pour CardSoft's resources into trying to do just that, launching a multi-front sales-and-marketing blitz to promote its product. CardSoft spent about $8 million to create and spearhead a payment-industry group called the Small Terminal Interoperability Platform ("STIP") consortium, A18,109—the "interoperability platform" being, of course, CardScript (by then called the "eAppliance Solution Suite") and its accompanying virtualization software, A17,571-73. CardSoft's goal was to create an "open system" "to solve this problem of interoperability" by producing a sort of industry "standard" à la STIP. A17,021-23. Virtually all of the major industry players and manufacturers—including VeriFone, Hypercom, and Ingenico—sat in during STIP's early meetings. A17,022-26, 17,616. Ogilvy also met personally with VeriFone and Hypercom executives urging them to adopt CardScript. A17,024-26.

The blitz was a bust. The manufacturers that attended the STIP meetings expressed no interest in shifting from their existing customized approach to CardSoft's. And market movers like Visa and other credit and debit card issuers were not forcefully urging it, either. A18,112. As much as CardSoft pushed, the company could not use STIP to convince manufacturers to incorporate CardScript into their terminals. A18,109. Separate one-on-one meetings also fell flat. A13,016, 17,037-38.

Multiple factors motivated manufacturers to keep their distance. CardScript would undermine their business model, which depended on customized solutions. A17,031, 17,512. CardScript would make hacking easier because when "one application … can suddenly run on … 10-plus-million devices," "all a hacker has to do is find one flaw, and all of a sudden, he has access to 10-million-plus devices." A17,525. And, even solely as it relates to programming effort, CardScript would not achieve the gains that Ogilvy envisioned because most merchants ask for highly individualized applications. A17,523-24, 17,631-32.

Additionally, installing CardScript on terminals would slow them down. This, too, would be a serious problem, as anyone who has ever

had to wait in a long checkout line can attest. In conventional terminals, application instructions communicate directly with the operating system and processor. A119, col. 9:41-45; A17,620. But, in CardScript-enabled terminals, they would have to pass through an extra layer—the CardScript virtual machine—to gain access to the underlying terminal. A17,620.

With its product a total failure in the market, CardSoft was reeling. The company, which had never been profitable, A17,027, now was hemorrhaging so much money as a result of its failed marketing push that it was fast approaching bankruptcy. As one executive bluntly put it, "the company was worth zero." A13,079-80. As far as the CardSoft board of directors was concerned, CardScript's market failure was Ogilvy's last straw. In 2001, CardSoft fired Ogilvy and brought in a new CEO, Don Sweet. *See* A17,027-28, 17,568.

But as Sweet told the rest of the CardSoft executives, "the basic truth[]" is that "[t]here is no market demand for [CardScript]." A13,072. As before, "Ingenico[,] Veri[F]one … and Hypercom [we]re totally against STIP," he recognized. A13,131. Unable to sell CardScript, and recognizing that "CardSoft cannot force [it] on the

market place," A13,088, Sweet tried to sell the company instead. But Intellectual Ventures, Sweet's only prospect and a well-known intellectual-property acquirer, rejected CardSoft's asking price of $4 million plus ten percent of any future royalties out of hand, stating that it was "clearly an order of magnitude or more apart in valuation" from how much CardSoft was actually worth. A13,068, 17,587, 17,590.

In 2007, with CardSoft stuck in the red and rapidly approaching insolvency, the board of directors threw in the towel. They assigned the company's assets in trust to a group of its creditors and called it a day. A13,115-16, 17,933-34.

CardSoft Obtains A Large Award From The Terminal Manufacturers Based On An Allegedly "Common Platform"

On March 6, 2008, the creditors caused CardSoft to sue nearly every significant payment terminal manufacturer—including VeriFone, Hypercom, and Ingenico—for allegedly infringing its patents. CardSoft's theory was broad. It claimed that the manufacturers were incorporating CardScript-style software into numerous terminals, running it on the terminals' underlying hardware and "underlying secure operating system[s]" such that "**[a]ll of the accused … devices**

in this case share [a] common platform, regardless of which Defendant manufactured and/or sold [them].” A616 (emphasis added).

The parties consented to have a magistrate judge conduct all proceedings in this case—including trial and entry of judgment—pursuant to 28 U.S.C. § 636(c). A472. Magistrate Judge Everingham was initially assigned to the case, A473, but he retired shortly after issuing the claim construction order, at which point Magistrate Judge Payne took over, A1331-32.

Claim Construction of “Virtual Machine.” Both CardSoft and the manufacturers agreed that “a virtual machine” is a “computer programmed to emulate a hypothetical computer.” A42. As the magistrate noted, the “parties’ only dispute concerning the term is whether the claimed virtual machine must ‘process instructions expressed in a hardware/operating system-independent language,’” as the manufacturers argued. *Id.*

The magistrate declined to construe “virtual machine” to require applications written in a language that was not specific to any particular operating system or hardware configuration. He reasoned that because “both the message processor and the function processor”

"*are part of the virtual machine"* and "can be implemented in the native software code of the microprocessor," then the applications "do not have to be expressed in 'a hardware/operating system-independent language' as Defendants' proposed construction would require." A46 (emphasis added). Similarly, the magistrate thought that such a requirement "runs contrary to the language of" two dependent claims that referred to the function processor and message processor "implemented in the native software code of the microprocessor." A46.

Trial. At trial, liberated by the claim construction ruling, CardSoft abandoned its original theory and did not argue that all of the accused devices share a common platform. A16,936. Instead, it argued that the manufacturers adopted "the technology in the patents, for their own systems" "in a way that made the terminals within their own company universal." *Id*. Specifically for VeriFone, it claimed that the operating systems had "sufficient commonality," A17,237, to make the terminals "effectively the same system," A17,233.

With respect to VeriFone and Hypercom, the jury awarded CardSoft $15.4 million in damages for infringement. A72. The jury found that Ingenico—whose terminal architecture was not materially

different from VeriFone's or Hypercom's but whose hospitality application, rather than operating system, was accused of being the virtual machine—had not infringed. A70; *see* A17,268-69.

VeriFone and Hypercom moved for judgment as a matter of law, arguing among other things that CardSoft's evidence was insufficient to satisfy the "virtual machine" limitation. A12,410-11, 12,432-33. Magistrate Judge Payne denied the motions, concluding that CardSoft offered, "in short form," evidence of a virtual machine. A82, 97. The magistrate then granted in part CardSoft's motion for an ongoing royalty. A13,014-15. He ordered the parties to negotiate an ongoing royalty, and severed the royalty claim into a separate cause of action. A13,015.

SUMMARY OF ARGUMENT

This Court has already held that the ordinary meaning of "virtual machine" is, as VeriFone and Hypercom urged below, software that runs applications that do not depend on any specific underlying operating system or hardware. *Nazomi Commc'ns, Inc. v. Arm Holdings, PLC*, 403 F.3d 1364, 1366 (Fed. Cir. 2005). Other courts agree. *See, e.g., United States v. Microsoft Corp.*, 253 F.3d 34 (D.C. Cir.

2001); *Sun Microsystems, Inc. v. Microsoft Corp.*, 87 F. Supp. 2d 992, 996 (N.D. Cal. 2000). And this ordinary understanding is consistent with the meaning of the famous Java Virtual Machine. *See, e.g., Oracle Am., Inc. v. Google Inc.*, No. C 10-03561, 2011 WL 1565988, at *1 (N.D. Cal. Apr. 27, 2011).

The patent specification confirms that the claimed "virtual machine" runs applications that do not depend on any specific underlying operating system or hardware. The patent explains that in conventional payment terminals the applications are written for specific payment terminals, A115, col. 2:67-3:2, and are generally not portable among terminals, A119, col. 9:48-50. The specification proposes a "virtual machine" to address that problem by writing the applications to be "readable by the virtual machine." A116, col. 3:34-39; A121, col. 14:37-39. The "virtual machine can be implemented on *any* hardware, BIOS/OS arrangement …. [It can] therefore facilitate[] portability of programs." A117, col. 5:1-3.

Despite the overwhelming indications from ordinary meaning and the specification, the magistrate judge declined to construe "virtual machine" to mean software that runs applications that do not depend

on any specific underlying operating system or hardware. Although the magistrate gave several reasons for rejecting VeriFone and Hypercom's proposed construction, each suffers from the same logical flaw: a failure to distinguish between the virtual machine itself and the applications written to run on the machine. The magistrate then compounded that fundamental mistake by misusing the doctrine of claim differentiation and applying the wrong legal standard to minimize the importance of the specification, errors that, even on their own, would further warrant reversal here.

Even accepting the magistrate judge's erroneous construction that the "virtual machine" need simply "emulate a hypothetical computer," and nothing more, CardSoft still failed to prove that VeriFone's and Hypercom's payment terminals infringed. For Hypercom, CardSoft simply did not present any evidence to show that the "virtual machine" limitation was met. With respect to VeriFone, CardSoft's expert's sole statement that "the Verix operating systems describe a hypothetical computer," A17,168, is a classic example of a generic and conclusory statement that this Court has rejected time and again as insufficient to support a verdict of infringement.

STANDARD OF REVIEW

This Court "review[s] claim construction *de novo* on appeal." *Cybor Corp. v. FAS Techs., Inc.*, 138 F.3d 1448, 1456 (Fed. Cir. 1998) (en banc).

This Court reviews rulings on motions for judgment as a matter of law under regional circuit law. *Harris Corp. v. Ericsson Inc.*, 417 F.3d 1241, 1248 (Fed. Cir. 2005). The United States Court of Appeals for the Fifth Circuit reviews such rulings *de novo*. *Mid-Continent Cas. Co. v. Eland Energy, Inc.*, 709 F.3d 515, 520 (5th Cir. 2013). The Court must reverse the ruling if it determines that "'there is no legally sufficient evidentiary basis for a reasonable jury to find as the jury did.'" *Id.* (quoting *Guile v. United States*, 422 F.3d 221, 225 (5th Cir. 2005)).

ARGUMENT

Patent claims capture "the subject matter which the applicant regards as [his] invention." 35 U.S.C. § 112(b). This Court construes disputed claim terms by examining "'their [ordinary] meaning in the field [of the invention],'" including "the context in which a term is used in the asserted claim," as well as "'the patent specification and the prosecution history.'" *Phillips v. AWH Corp.*, 415 F.3d 1303, 1313-14 (Fed. Cir. 2005) (en banc) (quoting *Multiform Desiccants, Inc. v.*

Medzam, Ltd., 133 F.3d 1473, 1477 (Fed. Cir. 1998)). "To prove infringement, the patentee must show that the accused device contains each limitation of the asserted claim." *Function Media, LLC v. Google Inc.*, 708 F.3d 1310, 1330 (Fed. Cir. 2013) (internal quotations omitted).

I. A "VIRTUAL MACHINE" IS SOFTWARE THAT RUNS APPLICATIONS THAT DO NOT DEPEND ON ANY SPECIFIC UNDERLYING OPERATING SYSTEM OR HARDWARE

This case largely turns on the proper interpretation of "virtual machine" as used in claim 1 of CardSoft's '945 patent. CardSoft proposed construing "virtual machine" to mean "a computer programmed to emulate a hypothetical computer for applications relating to transport of data." A42. VeriFone and Hypercom proposed construing "virtual machine" to mean a "computer programmed to emulate a hypothetical computer, which hypothetical computer processes *instructions expressed in a hardware/operating system-independent language*." *Id.* (emphasis added). By "instructions expressed in a hardware/operating system-independent language," VeriFone and Hypercom meant applications that do not depend on any specific underlying operating systems or hardware. A992, 995, 1018.

Magistrate Judge Everingham sided with CardSoft, concluding that the claimed virtual machine could run applications that depend on a specific underlying operating system or hardware. But that construction of "virtual machine" conflicts with the plain language, with the problem and proposed solution identified in the patent specification, and with CardSoft's position before the Patent Office. The magistrate judge reached this mistaken construction by conflating the virtual machine software with the applications that the virtual machine runs. This Court should reverse the erroneous construction.

A. The Ordinary Meaning Of "Virtual Machine" Refers To Software That Runs Applications That Do Not Depend On Any Specific Underlying Operating System Or Hardware

This Court "always" "begin[s] [its] construction with the words of the claim." *Adv. Cardiovascular Sys. Inc. v. Medtronic, Inc.*, 265 F.3d 1294, 1304 (Fed. Cir. 2001) (citing *Interactive Gift Express, Inc. v. Compuserve Inc.*, 256 F.3d 1323, 1331 (Fed. Cir. 2001)); *see Phillips*, 415 F.3d at 1312 ("We look to the words of the claims themselves"). Those words "'are generally given their ordinary and customary meaning.'" *Phillips*, 415 F.3d at 1312 (quoting *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582 (Fed. Cir. 1996)).

The Court has already held that the ordinary meaning of "virtual machine" is, as VeriFone and Hypercom urged below, software that can run applications that do not depend on any specific underlying operating system or hardware. In *Nazomi Communications, Inc. v. Arm Holdings, PLC*, this Court explained that in a virtual machine environment, a programmer writes programs in source code which "is then compiled into … bytecodes, 'instructions that look like machine code, *but aren't specific to any processor*.'" 403 F.3d at 1366 (emphasis added). A "Virtual Machine," the Court explained, is then used to "interpret[]" or "translat[e]" these instructions into something that the underlying processor can understand. *Id.* In remanding for further claim construction on what qualified as "instructions," this Court noted that the trial court needed to take into account the patent's statement that "*generic* instructions, such as bytecodes, indicate the operation of a virtual machine." *Id.* at 1369 (emphasis added) (citations omitted). *See also Nazomi Cmmc'ns, Inc. v. Nokia Corp.*, 739 F.3d 1339, 1340 (Fed. Cir. 2014) (noting that programs that are run on the "Java Virtual Machine" are not "processor-specific," and therefore can be "run on any platform and any operating system").

The D.C. Circuit has laid out a similar understanding of virtual machine. *See United States v. Microsoft Corp.*, 253 F.3d 34 (D.C. Cir. 2001). That court emphasized the "cross-platform" nature of the virtual machine technology at issue and explained how applications that are written without regard to the underlying hardware can run on different computer platforms because a virtual machine "translates" those applications "into instructions to the operating system." *Id.* at 74.

Other cases offer variations on this same critical theme. *See Oracle Am., Inc. v. Google Inc.*, No. C 10-03561, 2011 WL 1565988, at *1 (N.D. Cal. Apr. 27, 2011) (noting that normally an application "had to be written specifically for the hardware architecture that ultimately would run" it, but "a program written for [a] virtual machine could be run on any computer with [that] virtual machine, regardless of that computer's underlying hardware architecture"); *Sun Microsystems, Inc. v. Microsoft Corp.*, 999 F. Supp. 1301, 1302 (N.D. Cal. 1998) (explaining that "[m]ost computer systems implement a platform-dependent programming environment," but an application written for a virtual machine is "standardized" to enable portability among incompatible platforms).

In construing a claim term, a court should give effect to "the meaning that the term would have to a person of ordinary skill in the art" relevant to the invention at issue "at the time of the invention." *Phillips,* 415 F.3d at 1313 (citing cases). In 1997—when Ogilvy filed the Australian patent application to which the '945 patent claimed priority—the Java Virtual Machine was larger than life. In the year leading up to the patent application, the Java Virtual Machine "ha[d] perhaps received more press … than any other product since Windows 95." A13,271. "Forward-looking engineers [we]re rearchitecting entire networked systems, placing Java at the core." *Id.*

"[T]he main attraction" with the Java Virtual Machine "is the fact that Java applications are completely portable." A13,271. That is, "[w]rite your code once and you never need to port or even recompile it." *Id.* This is because when the Java application is first compiled, "[r]ather than producing machine-specific instructions, the Java compiler produces vendor-neutral bytecode." *Id.* Or, to put it another way, the Java Virtual Machine reads "bytecode instructions which have nothing to do with a particular computer architecture." James Golsing,

The Java Language: An Overview, Dartmouth Coll. (White Paper), Feb. 1995, at 5, *available at* http://www.cs.dartmouth.edu/~mckeeman/cs118/references/OriginalJavaWhitepaper.pdf. The Java Virtual Machine "then translates the bytecode into actual machine-specific instructions" for the computer to understand. A13,271.

By the time Ogilvy filed his application, engineers in Europe already were applying virtual machine technology to payment systems, constructing an Open Terminal Architecture ("OTA") that did exactly what CardSoft claimed to have invented with CardScript. The OTA, which was developed in 1996, was designed to make it "possible for credit card issuers and acquirers to write application programs that will be completely platform independent, and run on all OTA-compliant kernels." Peter Johannes, et al., *The Europay Open Terminal Architecture: A Forth-based Token System for Payment Terminals*, *in* Open Systems 27, 27 (The Institute for Applied Forth Research, Inc. 1996). *See* A4116. This is possible because the [redacted] A3996. Thus, as with CardSoft's claimed virtual machine, the [redacted]

Confidential Material Omitted

[redacted] A3999.[6]

Here, Ogilvy himself made clear that the '945 patent used the term "virtual machine" in exactly this way. The inventor explained that the applications that run on his virtual machine are not "compiled into the native code of the processor," that is, they are not written for specific terminals. A17,047. Instead, the applications are developed to "run on the virtual machine." A17,041.

A person of "ordinary skill in the art"—which Ogilvy clearly was—"is deemed to read the words used … with an understanding of their meaning in the field." *Phillips*, 415 F.3d at 1313 (quoting *Multiform Desiccants, Inc. v. Medzam, Ltd.*, 133 F.3d 1473, 1477 (Fed. Cir. 1998)). And to anyone in the field of computer science, the term "virtual machine" would have conjured software that runs applications that do not depend on any specific underlying operating system or hardware as

[6] VeriFone and Hypercom did not discover this OTA prior art until after they had filed invalidity contentions, and the magistrate judge denied them leave to supplement their invalidity contentions under Patent Local Rule 3.6. Because "[d]ecisions enforcing local rules in patent cases" are rarely reversed, *O2 Micro Int'l Ltd v. Monolithic Power Sys., Inc.*, 467 F.3d 1355, 1366-67 (Fed. Cir. 2006), VeriFone and Hypercom are not appealing that decision.

surely as the term "automobile" would have brought to mind a vehicle with four wheels.

B. The Specification Confirms That "Virtual Machine" Refers To Software That Runs Applications That Do Not Depend On Any Specific Underlying Operating System Or Hardware

The specification "'is the single best guide to the meaning of a disputed term.'" *Phillips*, 415 F.3d at 1315 (quoting *Vitronics*, 90 F.3d at 1582). This Court construes claims consistently with "'the scope of the invention'" as described in the written specification, *S3 Inc. v. nVidia Corp.*, 259 F.3d 1364, 1367 (Fed. Cir. 2001) (quoting *Miles Labs., Inc. v. Shandon Inc.*, 997 F.2d 870, 875 (Fed. Cir. 1993)), ensuring in particular that they are "'sufficient to achieve the fundamental purpose of the invention,'" *Honeywell Int'l, Inc. v. United States*, 609 F.3d 1292, 1302 (Fed. Cir. 2010) (quoting *Minn. Mining & Mfg. Co. v. Johnson & Johnson Orthopaedics, Inc.*, 976 F.2d 1559, 1567 (Fed. Cir. 1992)). The specification here indisputably confirms that the claimed "virtual machine" runs applications that do not depend on any specific underlying operating system or hardware.

The specification begins by explaining the problem that the invention overcomes: that in conventional payment terminals the

applications are written for specific payment terminals, A115-16, col. 2:67-3:2, and are generally not portable among terminals, A119, col. 9:48-50. Instead, "it is necessary to write a [new] program"—or, put differently, a new version of that same program—"specifically for each type of device." *Id.*, col. 9:50-51. In other words, "the programming will be different for different types of devices." A116, col. 3:6-7.

Over and over again, the specification discusses this core "portability" problem that exists in conventional terminals and that Ogilvy had intended to solve. It laments that "[b]ecause of the different hardware/software architectures" of payment terminals, A115, col. 2:34-35, "a new application must be created" for each type of payment terminal every time anyone wants to alter a single operation across any of their terminals, A116, col. 3:5-6. It notes that because programs are not portable among payment terminals, "the terminal owner is thus tied to the particular supplier/manufacturer of the particular brand of payment terminal." A115, col. 2:40-42. That, it observes, "causes cost, time and trouble when any operational alterations are required." *Id.*, col. 2:53-55. It emphasizes the breadth of the problem, given that "[t]here are many different brands of payment terminal" which "utilis[e]

many different software and hardware arrangements." *Id.*, col. 2:4-6. And the specification succinctly sums up the crux of all of these perceived drawbacks: "The programming alterations are not 'portable' between different types of devices." A116, col. 3:13-14.

Ogilvy proposed a "virtual machine" to address the problem caused by the "conventional" approach. A119, col. 9:38. The specification explains that the claimed virtual machine is "programmed to emulate a hypothetical [terminal]" and all terminals running it "emulate the same hypothetical [terminal]." A116, col. 3:40-43. And because the "virtual machine can be implemented on *any* hardware, BIOS/OS arrangement," it "therefore facilitates portability of programs." A117, col. 5:1-3 (emphasis added). A programmer looking to write a payment terminal application for those terminals thus need not create separate versions custom-tailored to work with each terminal's underlying system. Instead, the programmer writes only one version: the generic-language—or CardScript—version, which is the version "for the virtual [terminal]." A116, col. 3:45; A123, col. 17:39-41. The virtual machine then runs the application's terminal-independent instructions and translates them into commands that the underlying

Case: 14-1135 Document: 28 Page: 59 Filed: 02/18/2014

system can understand and act upon, regardless of its particular operating system/hardware configuration. It thus "creates a complete portable environment for program operations." A116, col. 3:45-46.

The specification emphasizes that "[d]ifferent *incompatible* computers may be programmed to emulate the same hypothetical computer." *Id.*, col. 3:41-43 (emphasis added). It does not matter if the computers are incompatible because of the hardware, or incompatible because of the operating system, or both. "The invention can be applied in *any* device." A117, col. 5:33-34. "Any computer programmed to emulate the hypothetical computer will thus be capable of executing *programs for the virtual computer*," thus "allow[ing] programs to operate *independent* of [the hardware] processor." A116, col. 3:36-45 (emphasis added). As a result, "[i]mplementation of such a virtual machine on payment terminal devices of different brands enables operation of the payment terminal devices or brands to be altered *merely by altering application commands generic to all brands*," because "[e]ach brand is seen by the application as the same virtual machine." A117, col. 5:4-9 (emphasis added).

By identifying the problem as the need to write "different" programming for "different types of devices," A116, col. 3:6-7, and proposing instead software that creates a "complete portable environment for program operations," *id.*, col. 3:45-46, the specification uses the term "virtual machine" in the same ordinary fashion the term was used in *Nazomi*, *Microsoft*, the Java Virtual Machine, and all the other ordinary uses discussed above: a virtual machine runs applications that do not depend on any specific underlying operating system or hardware.

C. CardSoft's Position During Patent Prosecution Confirms That "Virtual Machine" Refers To Software That Runs Applications That Do Not Depend On Any Specific Underlying Operating System Or Hardware

A patent's "prosecution history can often inform the meaning of the claim language" by "provid[ing] evidence of how the PTO and the inventor understood the patent." *Phillips*, 415 F.3d at 1317; *see also Vitronics*, 90 F.3d at 1582-83. Here, CardSoft's position during prosecution confirms that the claimed "virtual machine," like the virtual machine discussed in the specification, runs "instructions" that do not depend on any specific type of underlying terminal architecture.

The original point of contention between CardSoft and the PTO was how closely the claimed virtual machine resembled the Java Virtual Machine. CardSoft's position was not (as it is now) that its invention was nothing like the Java Virtual Machine because it does not necessarily run programs that are independent of the hardware and operating system. Rather, its position was that its software does everything that the Java Virtual Machine does—but takes it one step further.

Specifically, the focus was on the virtual machine of the Stern '249 patent. *See, e.g.*, A18,839-46, 18,847-51, 18,854-60, 18,862-63. The Stern virtual machine was a conventional Java Virtual Machine. A18,943, col. 5:30-34. All agree it processes generic-language (Java) instructions—instructions that "are not hardware specific." *Id.*, col. 5:34-40. CardSoft acknowledged during prosecution that "[o]ne important feature of the Java language is that it can be interpreted by a Java Virtual Machine." A1155. "Thus, a program written in Java language may run on a variety of computers each having incompatible hardware or operating systems, and each running a Java Virtual Machine." *Id.*

CardSoft conceded that, like the virtual machine of Stern, its claimed virtual machine is used so that "different incompatible computers (incompatible hardware and operating systems) may be programmed to emulate the same hypothetical computer." A1116. CardSoft went on to explain that like the Java Virtual Machine, the "[a]pplications may then be written for the hypothetical computer," not the underlying terminal, and are "therefore portable to the previously incompatible computers." *Id.*

The "significant[] differen[ce]," CardSoft pointed out, between the "Java Virtual Machine of Stern" and its claimed virtual machine is that CardSoft's virtual machine "includes a dedicated virtual message processor, which function is to [sic] perform *generic* handling of messages." A1155 (emphasis added). That is, CardSoft's virtual machine describes "an addition to a conventional virtual machine," a "virtual message processor." A1116. CardSoft maintained that the "Java Virtual Machine does not include such claimed dedicated virtual message processor." A1155. Thus, the difference between the two virtual machines is that CardSoft's virtual machine has a "dedicated virtual message processor" whereas the Java Virtual Machine does not.

Id.; A1116. But both virtual machines run applications that are written for the respective virtual machine rather than written for a specific operating system and hardware.

As with the ordinary meaning of virtual machine and the specification's discussion of the problem and solution, the prosecution history illustrates that CardSoft's claimed virtual machine runs "generic" applications, i.e., applications that do not depend on any specific underlying operating system or hardware.

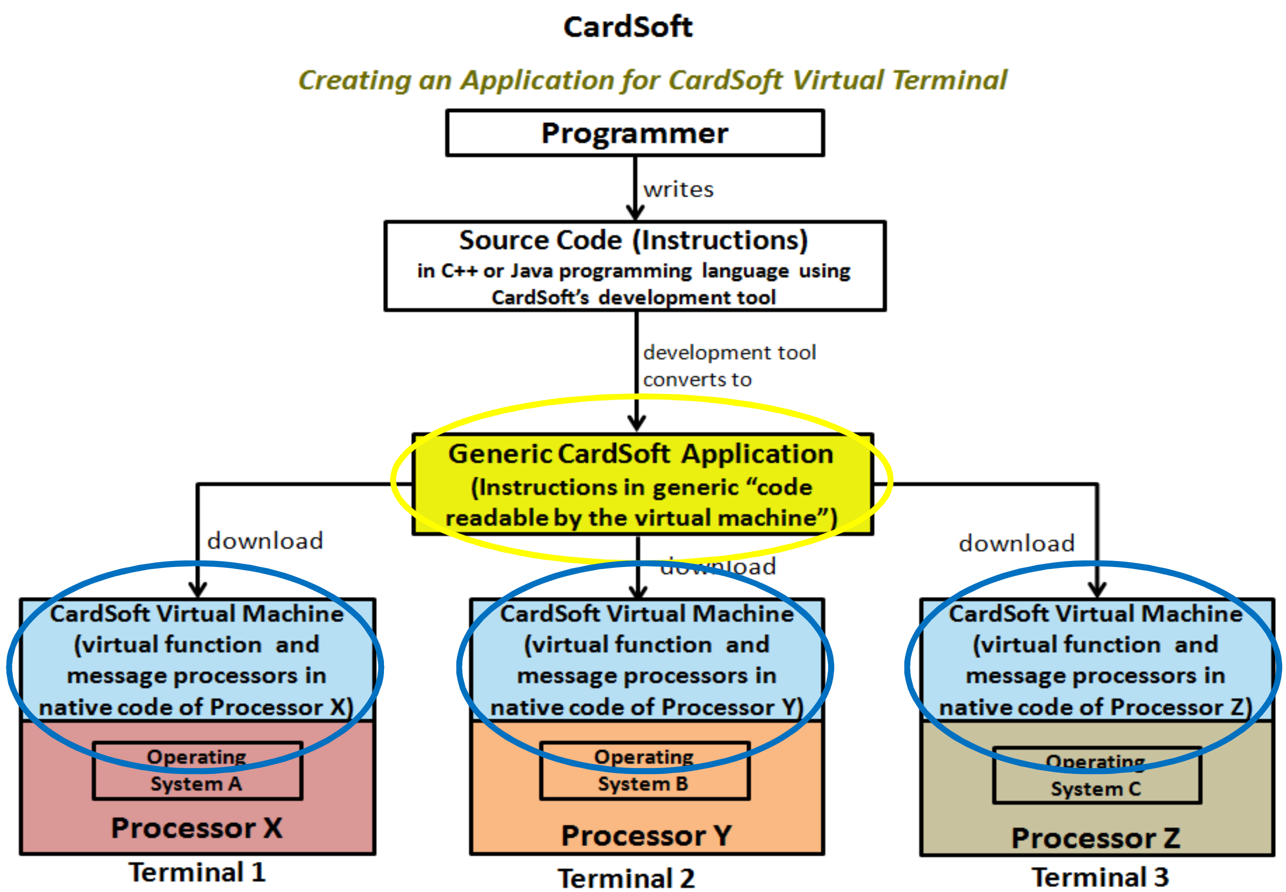
D. The Magistrate Judge's Reasons For Concluding Otherwise Were Fatally Flawed

This is the rare claim construction case where the cause of the error below is easy to discern. Although the magistrate gave several reasons for rejecting VeriFone and Hypercom's proposed construction, each suffers from the same simple logical error: a failure to distinguish between the virtual machine itself and the applications written to run on the virtual machine. The magistrate judge then compounded that fundamental mistake by misusing the doctrine of claim differentiation and applying the wrong legal standard to minimize the importance of the specification, errors that, even on their own, would warrant reversal here.

Confusing the applications with the virtual machine.

Magistrate Judge Everingham's most basic error was also his most glaring. He confused the applications that are readable by the virtual machine with parts of the virtual machine itself. Given that the whole purpose of the virtual machine is to convert generic application instructions into commands that the underlying system can handle, it is unsurprising that the virtual machine itself, which acts as an interface between those applications and the terminal, may be written in and communicate with the underlying terminal in terminal-dependent language. The magistrate's critical mistake in conflating the applications with the processors allowed him to conclude that the generic applications could, like the virtual machine processors, also be written in platform-specific code.

In terms of the earlier schematic illustration of CardSoft, the magistrate judge confused the application instructions circled in yellow with the virtual machine processors circled in blue:



More specifically, the magistrate confused the "instructions" with the "virtual … processors." There is no dispute that the portions of the virtual machine referred to as the "processors" do depend on the specific underlying operating system or hardware. A1174 (CardSoft stating that its "virtual machine may nonetheless be compiled into the specific language of a particular processor"). The dispute is whether the "instructions," i.e., the generic applications that the virtual machine runs, are dependent on the specific underlying operating system or hardware. Nevertheless, time and again, the magistrate failed to

distinguish between the virtual machine "processors" and the "instructions" that run on the virtual machine.

At the outset, when discussing the plain meaning of "virtual machine" and whether the virtual machine must "process instructions expressed in a hardware/operating system-independent language," the magistrate judge wrote that the manufacturers' "proposed limitation runs contrary to the language of the claims." A45-46. The magistrate then cited claim 5, noting that the "message processor is implemented in the native software code of the microprocessor in the device," and cited claim 6, noting that it says "the same for the function processor." A46. From these claims, the magistrate made the undisputed point that "both the [virtual] message processor and the [virtual] function processor, which are part of the virtual machine, can be implemented in the native software code of the microprocessor." *Id.*

The magistrate judge then relied on claims 5 and 6 to reject the view that the "instructions" must not depend on any specific underlying operating system or hardware. Claims 5 and 6, however, do not refer to instructions. They refer to the virtual machine's message and function

processors and state that those processors are implemented as "native software code of the microprocessor." A140, col. 51:18-25.

Nevertheless, citing these claims, the court then wrote:

> If both the message processor and the function processor, which *are part of the virtual machine*, can be implemented in the native software code of the microprocessor, then *they* do not have to be expressed in "a hardware/operating system-independent language" as Defendants' proposed construction would require.

A46 (emphasis added).

The quoted language confuses the virtual processors with the applications that the processors run. That the virtual processors "can be implemented in the native software code of the microprocessor" is not disputed, but says nothing about whether the applications that the virtual processors implement depend on a specific operating system or hardware configuration. The manufacturers' construction of "virtual machine" is that the *program instructions* are portable even if the virtual machine software itself is written for a particular operating system or hardware configuration. Only by conflating the processors with the instructions could the magistrate reach the conclusion that claims 5 and 6 are useful here.

Magistrate Judge Everingham exhibited the same confusion with respect to the specification. For instance, the magistrate judge noted that, in support of their construction that the virtual machine must process "instructions" that do not depend on any specific underlying operating system or hardware, VeriFone and Hypercom had pointed out that the specification criticizes prior art for "requiring applications written in hardware-specific code since such applications would not be portable to different devices." A46 (citing A116, col. 3:37-54) (underline by Magistrate Everingham); *see also* A995. The magistrate dismissed that aspect of the specification on the ground that it "does not … discuss whether the *virtual machine itself* can be written in hardware-specific code." A46 (emphasis added). "[I]ndeed," the magistrate went on to note, "the cited portion is silent on the topic of the code used to *implement the claimed virtual machine*." *Id.* (emphasis added). Only by conflating the instructions with the machine itself could the magistrate dismiss parts of the specification that discuss applications by referring to parts that discuss the virtual machine.

Similarly, the magistrate went on to state that "none of the other specification language to which Defendants cite states that the virtual

machine, or any part thereof, must necessarily be written in a hardware/operating system independent language." A46 (citations omitted). But the language in which the virtual machine itself must be written is beside the point. The issue is whether the application instructions that run on the virtual machine must be written in an operating system or hardware independent language. And, as explained above, the specification is far from silent on that. Only by focusing on the language of the virtual machine itself, instead of the language of the applications, could the magistrate dismiss as irrelevant the mountain of evidence in the specification showing that the instructions must not depend on the operating system or hardware.

And, yet again, the magistrate made the same mistake with respect to the prosecution history. The magistrate dismissed the history on the ground that it "do[es] not make any … clear disclaimer of *virtual machines* written in hardware-specific code." A47 (emphasis added). But, again, the question is not whether the "virtual machines" are "written in hardware-specific code." The question is whether the portable applications must be written in platform-independent code. As

explained above, the prosecution history confirms that the applications are portable and thus are not tied to any particular platform.

Contrary to the assumption driving the magistrate's analysis, the patent is clear that the virtual machine processors and the application instructions are distinct. The specification explains that "a virtual machine … includes virtual machine processors," A119, col. 9:64-65, whereas the "application includes [the] instructions," *id.*, col. 10:40-43. The instructions are not part of the processors. Instead—like application instructions in a conventional machine—they communicate with the virtual machine by telling the virtual machine what to do. *Id.*, col. 10:42-47.

Indeed, unlike the virtual processors, the specification notes that the instructions "*never* require[] translation to any real hardware processor." A116, col. 4:8-11, 34-37; A117, col. 5:22-25 (emphasis added). This is because the instructions are "to be expressed *only* in the language defined by the" virtual processors. A116, col. 4:9-10, 35-36; A117, col. 5:22-25 (emphasis added). "This allows programs to operate independent of [the underlying] processor." A116, col. 3:36-37; *accord id.*, col. 4:5-11 (while the "message processor means is preferably

translated into the native code of the microprocessor," the "message processor instructions are preferably virtual instructions to be expressed only in the language defined by the message processor means—and thus *never* requiring translation to any real hardware processor") (emphasis added); *id.*, col. 4:31-37 (same re "protocol processor means" and "protocol processor instructions"); A117, col. 5:19-25 (same re "function processor means" and "function processor instructions").

Figure 2 from the specification, reproduced below, illustrates that the "instructions" (highlighted in yellow) are separate from the "VM processors" (highlighted in blue).

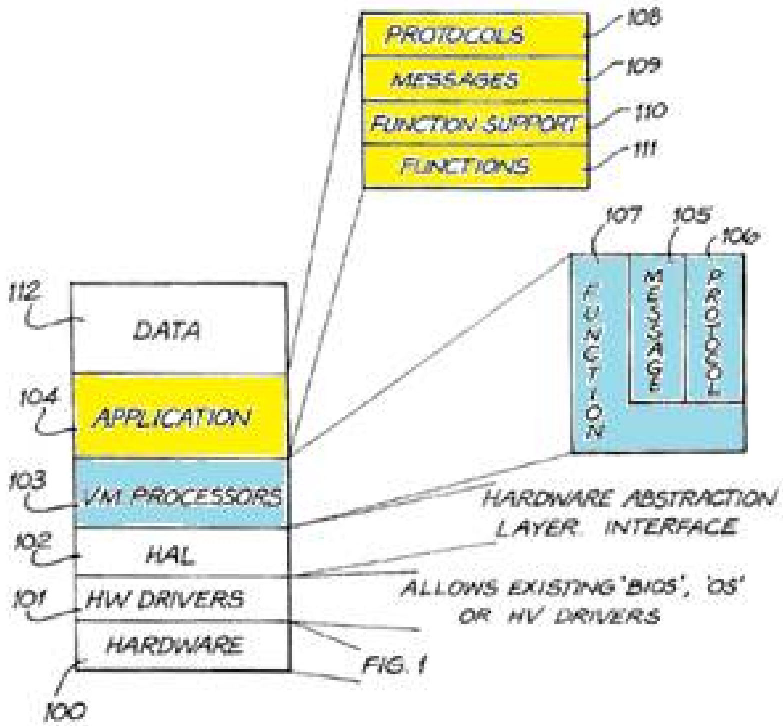


FIG. 2

A104, fig. 2 (highlighting added).

As the specification explains, Figure 2 makes clear that the "virtual machine processors **[103]** include a message processor **105** and a protocol processor **106**," A119, col. 10:18-19, highlighted in blue above. "The virtual machine processors **103** also comprise a function processor **107**." *Id.*, col. 10:34-35. Together, these "virtual machine processors," combined with the prior art HAL (102) and HW Drivers (103), make up the "virtual machine." *Id.*, col. 9:63-65. The "application **104** includes protocol instructions **108**, message instructions, **109**, function support

110 and function instructions **111**," *id.*, col. 10:40-42, highlighted in yellow above. "Application **104 15** controls the virtual machine **101**, **102**, **103** which in turn controls operation of the hardware **100**." *Id.*, col. 9:67-10:2.

In short, the specification establishes that the virtual machine is distinct from the applications it runs.

Misusing claim differentiation. The magistrate judge's invocation of claim differentiation changes none of this. Later in his opinion, when addressing the "instruction[s]" limitations, he observed that claim 7 and claim 8 of the '945 patent recite that "instruction[s] … do not require translation to the native software code of the microprocessor.'" A56, 58-59. This, he reasoned, "creates a presumption that Claim 1," which has no such requirement, must cover both instructions that do not require translation to the native software code of the microprocessor and instructions that do. A56, 58-59. And he was "not convinced that Defendants have overcome this presumption" with respect to either term. A57. He therefore concluded that the instructions processed by the virtual machine need not be in a hardware/operating system-independent language. A56-57, 59.

But this is wrong. The law of this Circuit is clear that while the doctrine of claim differentiation may create a presumption that each claim has a different scope, it creates "'only … a presumption,'" and a weak one at that. *Seachange Int'l, Inc. v. C-COR Inc.*, 413 F.3d 1361, 1369 (Fed. Cir. 2005) (quoting *Kraft Foods, Inc. v. Int'l Trading Co.*, 203 F.3d 1362, 1365-69 (Fed. Cir. 2000)). Here, as outlined in detail above, the claim language, specification, and prosecution history overcome this presumption.

Moreover, when construing the "means" language of a claim, such as the "message instruction means" of claim 7, courts "must look to the specification and interpret that language in light of the corresponding structure, material, or acts described therein." *In re Donaldson Co.*, 16 F.3d 1189, 1193 (Fed. Cir. 1994) (en banc); *id.* at 1195, 1196 (looking at the "Summary of the Invention" and the "preferred embodiment" to apply a limitation). The specification is clear that the "message processor instructions" "*never* require[] translation to any real hardware processor." A116, col. 4:10-11 (emphasis added). Magistrate Everingham therefore erred when he thought "[i]t is improper for the court to read such an embodiment into the claims" and dismissed it as

"merely a[n] embodiment of the claimed 'message processor instructions.'" A57.

Applying the wrong legal standard. Magistrate Everingham also erred when he stated that the specification did "not compel Defendants' proposed limitation" and noted that he was not convinced "the patentee clearly limited the scope" of the claimed invention. A45-46. By asking whether a limitation was compelled, the magistrate applied the outmoded approach from *Texas Digital Systems, Inc. v. Telegenix, Inc.*, 308 F.3d 1193 (Fed. Cir. 2002), that this Court sitting en banc repudiated in *Phillips*. *Texas Digital* took the position that "unless *compelled* otherwise," the naked words of a claim govern. 308 F.3d at 1202 (emphasis added). The en banc Court in *Phillips*, however, rejected that rigid approach and criticized *Texas Digital* for "improperly restrict[ing] the role of the specification in claim construction." 415 F.3d at 1320. Instead, it prescribed a more holistic inquiry, wherein the specification and all other relevant intrinsic evidence are *always* considered as critical aspects of the scope of a claim. *Id.* at 1315. The magistrate here "improperly restrict[ed] the role of the specification" in exactly the way that this Court denounced. *Id.* at 1320.

These last two errors compounded the problem caused by confusing the virtual processors with the application instructions. When the magistrate failed to distinguish between the virtual processors and the applications, the result was a construction that treated the virtual machine the same as the instructions processed by the virtual machine. That construction undermines the entire point of the virtual machine because it eliminates the requirement of portable applications. Even if valid, neither *Texas Digital*-style construction nor heavy reliance on claim differentiation could overcome the plain import of the claim language, specification, and prosecution history. That the magistrate judge saw fit to invoke these disfavored doctrines confirms how mistaken the decision below is.

E. A Noninfringement Ruling Is Compelled

Under the proper construction of "virtual machine," a ruling of noninfringement is compelled. VeriFone's and Hypercom's devices can only run applications that have been compiled into the native code of their respective terminals' underlying processors. VeriFone's Senior Vice President of Global Marketing, Paul Rasori, testified that VeriFone's applications are written and compiled specifically for the

terminals' operating system and hardware configurations. A17,522-23. CardSoft's technical expert, Tipton Cole, admitted that in order for applications to work on different Hypercom terminals, they have to be "compiled to the native code of the processors." A17,227-29. He said the same thing with respect to VeriFone, conceding that the "source code for those application programs is actually written and compiled into the native code of the microprocessor." *Id.*

There is no evidence that VeriFone's and Hypercom's terminals include software that runs applications that do not depend on any specific underlying operating system or hardware. VeriFone and Hypercom are entitled to a judgment of noninfringment as a matter of law. *See Saffran v. Johnson & Johnson*, 712 F.3d 549, 563-64 (Fed. Cir. 2013) (concluding that the district court erred in its claim construction ruling, reversing the judgment of infringement, and holding that defendants are entitled to a judgment of noninfringement); *On Demand Mach. Corp. v. Ingram Indus., Inc.,* 442 F.3d 1331, 1345 (Fed. Cir. 2006) (same); *CVI/Beta Ventures, Inc. v. Tura LP*, 112 F.3d 1146, 1149 (Fed. Cir. 1997) (same).

II. THERE WAS INSUFFICIENT EVIDENCE OF INFRINGEMENT EVEN UNDER THE ERRONEOUS CLAIM CONSTRUCTION

Even applying Magistrate Judge Everingham's erroneous construction of "virtual machine," CardSoft *still* did not prove that VeriFone and Hypercom infringed. Magistrate Judge Everingham construed the "virtual machine" limitation as requiring proof of "a computer programmed to emulate a hypothetical computer for applications relating to transport of data." A48. But CardSoft put on no evidence at all that any of VeriFone and Hypercom's accused devices—let alone all of them—contained any sort of "hypothetical computer." The jury verdict cannot stand. *See ePlus, Inc. v. Lawson Software, Inc.*, 700 F.3d 509, 521-22 (Fed. Cir. 2012) (reversing jury verdict of infringement because plaintiff did not "offer any evidence that showed or even suggested" that one of the limitations was met); *Harris*, 417 F.3d at 1256-57 (reversing a jury verdict of infringement because there was not substantial evidence to support the verdict).

A. CardSoft Offered Only A Single Conclusory Sentence To Establish A "Hypothetical Computer"

CardSoft's affirmative case consisted of the testimony of a single witness, Tipton Cole, CardSoft's technical expert. At trial, however,

Cole never attempted to explain how any of VeriFone's or Hypercom's terminals "emulate a hypothetical computer." *See* A17,259-64. Instead, he started off his analysis with the "virtual function processor and function processor instructions," A17,152-59 (VeriFone); A17,174-76 (Hypercom), which are distinct limitations under the patent. He then went on to testify about the "message instruction means" and "virtual message processor" limitations, A17,159-67 (VeriFone); A17,176-84 (Hypercom), which, again, are distinct limitations. He then summarily stated in a single sentence that "the Verix operating systems"—which are only one subset of the accused VeriFone operating systems—"*describe a hypothetical computer*" and "run over multiple different types of processors." A17,167-68 (emphasis added). For Hypercom, Cole did not use the term "hypothetical computer" at all in his analysis. Instead, he just stated that the "source code shows … a virtual machine that does run on … incompatible hardware." A17,184-85.

That was the *entirety* of Cole's infringement analysis at trial. In their motions for judgment as a matter of law, VeriFone and Hypercom pointed out that CardSoft completely skipped over the "virtual machine" limitation, even as wrongly construed by Judge Everingham.

A12,405-06, 12,429; *see Function Media,* 708 F.3d at 1330 ("To prove infringement, the patentee must show that the accused device contains *each* limitation of the asserted claim.") (emphasis added) (internal quotation marks omitted). But Magistrate Judge Payne allowed the verdict anyway because, in his view, it was sufficient that Cole offered a "short form" opinion for the virtual machine limitation. A82, 97. Citing this Court's decision in *Symbol Techs., Inc. v. Opticon, Inc.*, 935 F.2d 1569, 1576 (Fed. Cir. 1991), the magistrate judge said that VeriFone and Hypercom forfeited any right to demand further proof by declining to "cross-examine [Cole] on the basis for his opinion." A82, 97.

Contrary to the magistrate judge's suggestion, VeriFone and Hypercom *did* cross-examine Cole. Cole admitted on cross-examination that he "didn't single [the virtual machine limitation] out." A17,264. Cole's dismissal of the "virtual machine" limitation is in stark contrast to the expert's analysis in *Symbol Technologies*. There, the expert, "[u]sing the exhibits as a guide," walked the jury through limitation-by-limitation as to why he believed the accused devices infringed. 935 F.2d at 1574. After the jury found infringement, this Court excused the expert's failure to verbalize every jot and tittle of the rationale

underpinning the exhibits, saying that if the defendant wanted more than the "summary testimony" given, he should have engaged in better cross-examination. *See id.* at 1575-76.

Cole's statement in this case contained nothing that even approaches what the *Symbol Technologies* Court characterized as sufficient "summary testimony." Instead, his testimony is a quintessential example of a "generic statement" that is "too conclusory to sustain [a] verdict" of infringement. *Krippelz v. Ford Motor Co.*, 667 F.3d 1261, 1269 (Fed. Cir. 2012). In *Krippelz*, the issue was whether a prior art teaches a required "conical beam of light." *Id.* at 1265. This Court held that the expert's "generic statements that the 'conical' limitation was unmet" were "too conclusory to sustain the jury's verdict." *Id.* at 1268-69. Similarly, this Court observed in another case that an expert's "testimony that the 'conductors' in the accused processes and the claimed processes were the 'same' and performed the 'same function' was merely generalized testimony as to overall similarity." *Texas Instruments Inc. v. Cypress Semiconductor Corp.*, 90 F.3d 1558, 1567-68 (Fed. Cir. 1996). It criticized the expert for not

providing "particularized testimony" and held that, without doing so, "his testimony cannot support a finding" of infringement. *Id.* at 1568.

Here too, Cole did not offer any "particularized testimony" to show how exactly VeriFone's and Hypercom's terminals emulate a hypothetical computer. With respect to Hypercom, Cole did not even mention the term at all, let alone explain how it could be found in Hypercom's devices. As to VeriFone, Cole made only the single bald statement that "the Verix operating systems describe a hypothetical computer." A17,168. And he said *nothing* on that point with respect to the other VeriFone operating systems that were also accused and for which "each limitation" likewise needed to be proved. *Bowers v. Baystate Techs., Inc.*, 320 F.3d 1317, 1334 (Fed. Cir. 2003). On the issue of whether there *is* a hypothetical computer, it is hard to get more "generic" and "conclusory" than that.

B. The Evidence Overwhelmingly Shows That VeriFone And Hypercom Terminals Are Actual, Not "Hypothetical," Computers

Cole's testimony that VeriFone's "operating systems form a single platform" making all the terminals "effectively the same system," A17,232-33, changes none of this. Even if that were true, it would do

nothing to establish that any of those terminals contains a hypothetical computer. Uncontested evidence at trial showed that the accused terminals are actual, not hypothetical, computers.

Rasori, for example, testified that VeriFone spent thousands of hours developing each operating system for a specific processor or processors. A17,503-07. He testified that "Verix was developed internally at VeriFone" in the mid-1990s for the Motorola 68000 CPU. A17,503. As "technology progresse[d]," VeriFone decided "to move to a next generation of a hardware platform," and "had to create a different operating system" to run on the ARM9, which it named "Verix V." A17,503-04. Likewise, the Verix eVo operating system, which also took "thousands of hours" to develop, could only work on the ARM 11 processor. A17,506-07. Rasori testified that "an application that's developed for a Verix eVo operating system" cannot work on a Verix V operating system. A17,507. So too, in order for an application that was written for the Verix operating system to work on the Verix V operating system, "[y]ou have to go into the source code, make the changes to the source code, and actually compile it to the specific operating system." A17,521.

Cole did not dispute any of this. Instead, the only evidence he offered to support his statement was a reference to the deposition testimony of a VeriFone witness, Dave Faoro, VeriFone's Chief Payment Security Officer. A17,151-52. Faoro had testified that the "fundamental operation" of the operating systems on the various terminals is the same. A17,069. By that, he explained, he meant that on a "really [] high level," they handle "the normal kind of switching and … time-slicing processing that a typical OS does," and on a "higher level would be interfaces to applications" and "provid[ing] other functionality … like specialized capabilities and functions to interface to the hardware or use the hardware properly." A17,071-72. In other words, on a "really high level," they do what all operating systems—including the Windows and Mac OSs—do.

But the mere fact that VeriFone's various operating systems on a "really high level" can handle "the normal kind of" functions that other operating systems can does not convert any of them into a hypothetical computer. Indeed, it has no bearing on the issue at all. Faoro's description of VeriFone's operating systems could be applied to *any* operating systems. Every operating system in some sense interfaces

with applications and controls the general functioning of the computer on which it is installed. By definition, "[a]n operating system is a control program. A control program manages the execution of user programs …." Abraham Silberschatz et al., *Operating System Concepts* 5 (6th ed. 2002).

Accepting CardSoft's theory in this case thus would implicate *any* properly functioning operating system that is part of a line of similar operating systems running on different computers. For example, Apple's OS X operating system, installed on Mac computers, and Apple's iOS 7, installed on iPhones, would qualify because they provide similar functionality on a "really high level." Even Windows and Mac operating systems might qualify, as both interface with applications and hardware and provide other "similar functionality" in important respects. If that is all it takes to show a "hypothetical computer," then practically any modern computer would potentially infringe the '945 patent. That cannot be right.

* * *

A jury verdict of infringement cannot stand when it is based on conclusory expert testimony or against the overwhelming weight of the

evidence. Here, even under the wrong claim construction, the verdict rests on a single conclusory sentence that is impossible to reconcile with the record evidence. Reversal is required.

CONCLUSION

This Court should reverse the judgment below and enter judgment in favor of VeriFone and Hypercom.

Dated: February 18, 2014

Respectfully submitted,

By: /s/ E. Joshua Rosenkranz
E. Joshua Rosenkranz
Mark S. Davies
Richard A. Bierschbach
Brian D. Ginsberg
Cam T. Phan
Orrick, Herrington & Sutcliffe LLP
51 West 52nd Street
New York, New York 10019
Telephone: (212) 506-5000
Facsimile: (212) 506-5151
jrosenkranz@orrick.com

Robert W. Kantner
Jones Day
2727 North Harwood Street
Dallas, TX 75201
(214) 220-3939
rwkanter@jonesday.com

Case: 14-1135 Document: 28 Page: 87 Filed: 02/18/2014

ADDENDUM

Memorandum Opinion and Order, Dated September 23, 2011

UNITED STATES DISTRICT COURT
EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

CARDSOFT, INC., ET AL.	§	
	§	
vs.	§	CASE NO. 2:08-CV-98-CE
	§	
VERIFONE HOLDINGS, INC., ET AL.	§	

<u>MEMORANDUM OPINION AND ORDER</u>

I. INTRODUCTION

Plaintiffs CardSoft (Assignment for the Benefit of Creditors) LLC and CardSoft, Inc. (collectively "CardSoft" or "Plaintiffs") brought this action against Verifone Holdings, Inc., Verifone Inc., Hypercom Corporation, Ingenico S.A., Ingenico Corp., Ingenico Inc., Way Systems, Inc., Shera International Ltd. and Blue Bamboo (USA), Inc.[1] (collectively "Defendants"), alleging infringement of CardSoft's U.S. Patent Nos. 6,934,945 ("the '945 Patent") and 7,302,683 ("the '683 Patent").[2] The court held a *Markman* hearing on July 29, 2011. After considering the submissions and the arguments of counsel, the court issues the following order concerning the parties' claim construction disputes.

II. THE PATENT-IN-SUIT

The patents-in-suit are entitled "Method and Apparatus for Controlling Communications" and are directed "to preparing and processing information to be communicated via a network or to or from other data carriers." '945 Patent at Abstract. The Abstract of the invention explains that:

---

[1] Defendants Shera International Ltd. and Blue Bamboo (USA), Inc. have been dismissed from this case. *See* Dkt. No. 226.

[2] The '945 and '683 Patent share a common specification, and therefore, for convenience purposes, all future citations will be to the specification of the '945 Patent.

> For implementation of a novel "virtual machine" of the present invention, a minimal amount of hardware is required. Prior art virtual machines tend to slow down operation of the device as they interface between an application program and device drivers. The novel virtual machine incorporates a virtual message processing means that is arranged to construct, deconstruct and compare messages and applied in the native code of the processor. The message instruction means directs and controls the message processor. Similarly, a protocol processor means governs and organs [sic] communications, under the direction of a protocol instruction means in the application. These elements of the novel virtual machine increase the speed and efficiency and allow implementation of a practical device for use in communications, able to be implemented on different hardware having different BIOS/OS.

Id. Claim 1 of the '945 Patent, which is representative of the claims of the patents-in-suit, recites:

> 1. A communication device which is arranged to process messages for communications, comprising a virtual machine means which includes
>
> > a virtual function processor and function processor instructions for controlling operation of the device, and
> >
> > message induction [sic] means including a set of descriptions of message data;
> >
> > a virtual message processor, which is arranged to be called by the function processor and which is arranged to carry out the message handling tasks of assembling the messages, disassembling messages and comparing the messages under the direction of the message instruction means that is arranged to provide directions for operation of the virtual message processor, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task,
> >
> > wherein the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems.

Id. at 50:49-67.

III. GENERAL PRINCIPLES GOVERNING CLAIM CONSTRUCTION

"A claim in a patent provides the metes and bounds of the right which the patent confers on the patentee to exclude others from making, using or selling the protected invention." *Burke, Inc. v. Bruno Indep. Living Aids, Inc.*, 183 F.3d 1334, 1340 (Fed. Cir. 1999). Claim construction

is an issue of law for the court to decide. *Markman v. Westview Instruments, Inc.*, 52 F.3d 967, 970-71 (Fed. Cir. 1995) (en banc), *aff'd*, 517 U.S. 370 (1996).

To ascertain the meaning of claims, the court looks to three primary sources: the claims, the specification, and the prosecution history. *Markman*, 52 F.3d at 979. The specification must contain a written description of the invention that enables one of ordinary skill in the art to make and use the invention. *Id.* A patent's claims must be read in view of the specification, of which they are a part. *Id.* For claim construction purposes, the description may act as a sort of dictionary, which explains the invention and may define terms used in the claims. *Id.* "One purpose for examining the specification is to determine if the patentee has limited the scope of the claims." *Watts v. XL Sys., Inc.*, 232 F.3d 877, 882 (Fed. Cir. 2000).

Nonetheless, it is the function of the claims, not the specification, to set forth the limits of the patentee's invention. Otherwise, there would be no need for claims. *SRI Int'l v. Matsushita Elec. Corp.*, 775 F.2d 1107, 1121 (Fed. Cir. 1985) (en banc). The patentee is free to be his own lexicographer, but any special definition given to a word must be clearly set forth in the specification. *Intellicall, Inc. v. Phonometrics, Inc.*, 952 F.2d 1384, 1388 (Fed. Cir. 1992). Although the specification may indicate that certain embodiments are preferred, particular embodiments appearing in the specification will not be read into the claims when the claim language is broader than the embodiments. *Electro Med. Sys., S.A. v. Cooper Life Sciences, Inc.*, 34 F.3d 1048, 1054 (Fed. Cir. 1994).

This court's claim construction decision must be informed by the Federal Circuit's decision in *Phillips v. AWH Corporation*, 415 F.3d 1303 (Fed. Cir. 2005) (en banc). In *Phillips*, the court set forth several guideposts that courts should follow when construing claims. In particular, the court reiterated that "the *claims* of a patent define the invention to which the

patentee is entitled the right to exclude." 415 F.3d at 1312 (emphasis added) (*quoting Innova/Pure Water, Inc. v. Safari Water Filtration Systems, Inc.*, 381 F.3d 1111, 1115 (Fed. Cir. 2004)). To that end, the words used in a claim are generally given their ordinary and customary meaning. *Id*. The ordinary and customary meaning of a claim term "is the meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention, i.e., as of the effective filing date of the patent application." *Id*. at 1313. This principle of patent law flows naturally from the recognition that inventors are usually persons who are skilled in the field of the invention and that patents are addressed to and intended to be read by others skilled in the particular art. *Id*.

The primacy of claim terms notwithstanding, *Phillips* made clear that "the person of ordinary skill in the art is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification." *Id*. Although the claims themselves may provide guidance as to the meaning of particular terms, those terms are part of "a fully integrated written instrument." *Id*. at 1315 (quoting *Markman*, 52 F.3d at 978). Thus, the *Phillips* court emphasized the specification as being the primary basis for construing the claims. *Id*. at 1314-17. As the Supreme Court stated long ago, "in case of doubt or ambiguity it is proper in all cases to refer back to the descriptive portions of the specification to aid in solving the doubt or in ascertaining the true intent and meaning of the language employed in the claims." *Bates v. Coe*, 98 U.S. 31, 38 (1878). In addressing the role of the specification, the *Phillips* court quoted with approval its earlier observations from *Renishaw PLC v. Marposs Societa' per Azioni*, 158 F.3d 1243, 1250 (Fed. Cir. 1998):

> Ultimately, the interpretation to be given a term can only be determined and

> confirmed with a full understanding of what the inventors actually invented and intended to envelop with the claim. The construction that stays true to the claim language and most naturally aligns with the patent's description of the invention will be, in the end, the correct construction.

Phillips, 415 F.3d at 1316. Consequently, *Phillips* emphasized the important role the specification plays in the claim construction process.

The prosecution history also continues to play an important role in claim interpretation. Like the specification, the prosecution history helps to demonstrate how the inventor and the PTO understood the patent. *Id.* at 1317. Because the file history, however, "represents an ongoing negotiation between the PTO and the applicant," it may lack the clarity of the specification and thus be less useful in claim construction proceedings. *Id*. Nevertheless, the prosecution history is intrinsic evidence that is relevant to the determination of how the inventor understood the invention and whether the inventor limited the invention during prosecution by narrowing the scope of the claims. *Id.*

Phillips rejected any claim construction approach that sacrificed the intrinsic record in favor of extrinsic evidence, such as dictionary definitions or expert testimony. The *en banc* court condemned the suggestion made by *Texas Digital Systems, Inc. v. Telegenix, Inc.*, 308 F.3d 1193 (Fed. Cir. 2002), that a court should discern the ordinary meaning of the claim terms (through dictionaries or otherwise) before resorting to the specification for certain limited purposes. *Phillips*, 415 F.3d at 1319-24. The approach suggested by *Texas Digital*—the assignment of a limited role to the specification—was rejected as inconsistent with decisions holding the specification to be the best guide to the meaning of a disputed term. *Id*. at 1320-21. According to *Phillips*, reliance on dictionary definitions at the expense of the specification had the effect of "focus[ing] the inquiry on the abstract meaning of words rather than on the meaning of claim

terms within the context of the patent." *Id*. at 1321. *Phillips* emphasized that the patent system is based on the proposition that the claims cover only the invented subject matter. *Id.* What is described in the claims flows from the statutory requirement imposed on the patentee to describe and particularly claim what he or she has invented. *Id*. The definitions found in dictionaries, however, often flow from the editors' objective of assembling all of the possible definitions for a word. *Id*. at 1321-22.

Phillips does not preclude all uses of dictionaries in claim construction proceedings. Instead, the court assigned dictionaries a role subordinate to the intrinsic record. In doing so, the court emphasized that claim construction issues are not resolved by any magic formula. The court did not impose any particular sequence of steps for a court to follow when it considers disputed claim language. *Id*. at 1323-25. Rather, *Phillips* held that a court must attach the appropriate weight to the intrinsic sources offered in support of a proposed claim construction, bearing in mind the general rule that the claims measure the scope of the patent grant.

The patents-in-suit include claim limitations that are alleged to fall within the scope of 35 U.S.C. § 112, ¶ 6. "An element in a claim for a combination may be expressed as a means or step for performing a specified function without the recital of structure. . . in support thereof, and such claim shall be construed to cover the corresponding structure . . . described in the specification and equivalents thereof." 35 U.S.C. § 112, ¶ 6. The first step in construing a means-plus-function limitation is to identify the recited function. *See Micro Chem., Inc. v. Great Plains Chem. Co.*, 194 F.3d 1250, 1258 (Fed. Cir. 1999). The second step in the analysis is to identify in the specification the structure corresponding to the recited function. *Id.* The "structure disclosed in the specification is 'corresponding' structure only if the specification or prosecution history clearly links or associates that structure to the function recited in the claim."

Medical Instrumentation and Diagnostics Corp. v. Elekta AB, 344 F.3d 1205, 1210 (Fed. Cir. 2003) (citing *B. Braun v. Abbott Labs*, 124 F.3d 1419, 1424 (Fed. Cir. 1997)). The patentee must clearly link or associate structure with the claimed function as part of the quid pro quo for allowing the patentee to express the claim in terms of function pursuant to § 112, ¶ 6. *See id.* at 1211; *see also Budde v. Harley-Davidson, Inc.*, 250 F.3d 1369, 1377 (Fed. Cir. 2001). The "price that must be paid" for use of means-plus-function claim language is the limitation of the claim to the means specified in the written description and equivalents thereof. *See O.I. Corp. v. Tekmar Co.*, 115 F.3d 1576, 1583 (Fed. Cir. 1997). "If the specification does not contain an adequate disclosure of the structure that corresponds to the claimed function, the patentee will have 'failed to particularly point out and distinctly claim the invention as required by the second paragraph of section 112,' which renders the claim invalid for indefiniteness." *Blackboard, Inc. v. Desire2Learn, Inc.*, 574 F.3d 1371, 1382 (Fed. Cir. 2009) (quoting *In re Donaldson Co.*, 16 F.3d 1189, 1195 (Fed. Cir. 1994) (en banc)). It is important to determine whether one of skill in the art would understand the specification itself to disclose the structure, not simply whether that person would be capable of implementing the structure. *See Atmel Corp. v. Info. Storage Devices, Inc.*, 198 F.3d 1374, 1382 (Fed. Cir. 1999); *Biomedino*, 490 F.3d at 953. Fundamentally, it is improper to look to the knowledge of one skilled in the art separate and apart from the disclosure of the patent. *See Medical Instrumentation*, 344 F.3d at 1211-12. "[A] challenge to a claim containing a means-plus-function limitation as lacking structural support requires a finding, by clear and convincing evidence, that the specification lacks disclosure of structure sufficient to be understood by one skilled in the art as being adequate to perform the recited function." *Budde*, 250 F.3d at 1376-77

IV. CLAIM TERMS IN DISPUTE

a. "virtual machine means"

Plaintiffs' Proposed Construction	Defendants' Proposed Construction
A computer programmed to emulate a hypothetical computer for applications relating to transport of data, including payment terminal devices in which message processing and communication comprise a significant proportion of the operation of the device.	A computer programmed to emulate a hypothetical computer, which hypothetical computer processes instructions expressed in a hardware/operating system-independent language on the communications device, including function processor instructions and message instructions.

Claim 1 of the '945 Patent, which is representative of the use of the term "virtual machine means" in the patents-in-suit, recites as follows:

> 1. A communication device which is arranged to process messages for communications, comprising a *virtual machine means* which includes
>
> a virtual function processor and function processor instructions for controlling operation of the device….

'945 Patent at 50:49-67 (emphasis added). CardSoft argues that the claimed "virtual machine means" should be construed to mean "a computer programmed to emulate a hypothetical computer for applications relating to transport of data, including payment terminal devices in which message processing and communication comprise a significant proportion of the operation of the device." Defendants, on the other hand, argue that the term should be construed to mean "a computer programmed to emulate a hypothetical computer, which hypothetical computer processes instructions expressed in a hardware/operating system-independent language on the communications device, including function processor instructions and message instructions." The parties' only dispute concerning the term is whether the claimed virtual machine must "process instructions expressed in a hardware/operating system-independent language."

In a theme that recurs throughout all of their proposed constructions, Defendants seek to limit the claimed virtual machine to a hypothetical computer that "processes instructions expressed in a hardware/operating system-independent language on the communications device…." Defendants argue that this proposed limitation is required by the language of the claims, the common specification's description of the "virtual machine," and the prosecution history of the patents-in-suit. Defendants note that all of the independent claims of the patents-in-suit require that the "virtual machine means" be "emulatable in different computers having incompatible hardwares or operating systems." *Id.* at 50:65-67; 52:13-15; 52:34-36; '683 Patent at 58:8-10. Thus, Defendants argue that the virtual machine's emulation of the hypothetical computer must somehow overcome incompatibility between both different operating systems and different hardware (processors) that can only understand and process its own specific native code. Defendants contend that the only way that the claimed "virtual machine means" can overcome these incompatibilities is if the virtual machine is programmed and receives instructions in a language that is independent of both the hardware processor and the operating system. Furthermore, Defendants argue that this conclusion is supported by the common specification, which consistently emphasizes the importance of the virtual machine and its components being independent of the specific hardware processor. *See, e.g., id.* at 2:3-3:8, 3:40-45, 5:4-8, 9:37-45, at 9:66-10:21, 17:24-47. For example, the common specification explains that:

> In conventional devices, each time a message is constructed or deconstructed, the operation of the machine will be handled by the application program. To change operation of the machine, the application must be changed. This is laborious, and gives rise to problems, as discussed above.
>
> The technique of creating a virtual processor (or in this case microprocessor) is well known and referred to as an interpreter. This allows programs to operate

> independent of [sic] processor. With the newer technique of also creating virtual peripherals then the whole is referred to as a "virtual machine".
>
> A virtual machine is computer programmed to emulate a hypothetical computer. Different incompatible computers may be programmed to emulate the same hypothetical computer. Any computer programmed to emulate the hypothetical computer will thus be capable of executing programs for the virtual computer. This creates a complete portable environment for program operations.
>
> A problem with virtual machines is emulation is slower than normal program execution. For some applications this performance penalty is a significant problem.
>
> The above problems and disadvantages which have been discussed specifically in relation to devices configured to process payment transactions also would apply to devices configured to prepare and process any information to be sent or received via a network, not restricted to payment transaction information.
>
> A virtual machine is computer programmed to emulate a hypothetical computer. Different incompatible computers may be programmed to emulate the same hypothetical computer. *Any computer programmed to emulate the hypothetical computer will thus be capable of executing programs for the virtual computer. This creates a complete portable environment for program operations*.

Id. at 3:40-46.

Defendants also contend that the during prosecution of the patents-in-suit, CardSoft made several clear disclaimers of claim scope by repeatedly stressing the importance of the virtual machine's compatibility and portability. First, Defendants argue that, in making the following statements, the applicant was explaining that the virtual machine of the patents-in-suit is coded using a language independent of both the hardware processor and the operating system of the device:

> As discussed in the Specification page 6, lines 2-3 of the present application, a virtual machine is a computer, which is programmed to emulate a hypothetical computer. *This means that different incompatible computers (incompatible hardware and operating systems) may be programmed to emulate the same hypothetical computer. Applications may then be written for the hypothetical computer, which are therefore portable to the previously incompatible computers….*

> The present invention … does not describe a conventional virtual machine, but an addition to a conventional virtual machine.… There is a conventional virtual machine processor, being the "function processor", which together with the HAL and the instructions to operate it ("primitives"), controls the overall operation of the communications device. In addition, however, a separate virtual processor, the virtual message processor, is provided, the specific function of which is to disassemble, assemble, and compare messages.
>
> *The virtual machine architecture of the present invention, therefore, is not conventional. It includes two virtual processors (and three in the preferred embodiment where a protocol processor is also provided).*

Ex. C at 3-4, attached to Defendants' Responsive Claim Construction Brief, Dkt. No. 210 (original emphasis omitted, emphasis added); *see also* Ex. D at 2-4 (applicant explaining that the prior art does not teach the claimed virtual machine that is portable and not dependent on particular hardware). Defendants also argue that the following prosecution history statements operate as a clear disclaimer of claim scope:

> Applicant respectfully points out that Stern fails to teach the claimed "virtual machine means" that is emulatable in different computers having incompatible hardware or operating systems." The cited Stern col 6, lines 18-23, describes merely JavaOS being operable on different processors supporting the Java Virtual Machine.
>
> The presently claimed virtual machine means *is not just a JavaOS or a Java Virtual Machine*. As recited in Claim 1 (now further amended), the claimed Virtual Machine Means comprises, inter alia, (1) the virtual function processor, (2) the message instruction means, and (3) the virtual message processor that performs several tasks, one of which being "comparing [of] the messages under the direction of the message instruction means that is arranged to provide directions for operation of the virtual message processor."

Ex G at 8-9, attached to Defendants' Responsive Claim Construction Brief, Dkt. No. 210 (emphasis added). Defendants contend that, since the applicant repeatedly argued that the virtual machine of the patents-in-suit eliminate dependence on the hardware of the device, the applicant clearly disavowed any claim scope where the virtual machine is dependent on the hardware.

Having carefully reviewed Defendants' arguments, the court is not convinced that the patentee clearly limited the scope of his invention to "virtual machines" that "process[]

instructions expressed in a hardware/operating system-independent language on the communications device, including function processor instructions and message instructions." First, Defendants' proposed limitation runs contrary to the language of the claims. For example, Claim 5 of the '945 Patent recites that the message processor is implemented in the native software code of the microprocessor in the device. *See* '945 Patent at 51:18-22. Furthermore, Claim 6 recites the same for the function processor. *See id*. at 51:23-25. If both the message processor and the function processor, which are part of the virtual machine, can be implemented in the native software code of the microprocessor, then they do not have to be expressed in "a hardware/operating system-independent language" as Defendants' proposed construction would require.

Second, the specification sections on which Defendants rely do not compel Defendants' proposed limitation. For example, column 3, lines 29-55 of the specification, which is quoted above, criticizes prior art virtual machines for requiring <u>applications</u> written in hardware-specific code since such applications would not be portable to different devices. '945 Patent at 3:37-54. It does not, however, discuss whether the virtual machine itself can be written in hardware-specific code – indeed, the cited portion is silent on the topic of the code used to implement the claimed virtual machine. Likewise, none of the other specification language to which Defendants cite states that the virtual machine, or any part thereof, must necessarily be written in a hardware/operating system independent language in order to be emulatable in different computers.

Finally, Defendants' contention that the doctrine of prosecution disclaimer supports their proposed limitation is rejected. For prosecution disclaimer to apply, there must be a clear and unequivocal disavowal of a particular construction or scope of a claim term. *See, e.g.*,

Honeywell Int'l, Inc. v. Universal Avionics Sys., 493 F.3d 1358 (Fed. Cir. 2007). The portions of the prosecution history cited and relied upon by Defendants, however, do not make any such clear disclaimer of virtual machines written in hardware-specific code. For example, Defendants allege that the applicant argued to the PTO that the claimed virtual machine was not conventional because it was coded using language independent of hardware. To the contrary, the passages on which Defendants rely demonstrates that the applicant argued that the claimed virtual machine was not conventional because "[i]t includes two virtual processors [the virtual message processor and the virtual function processor]... ." Ex. C at 3-4, attached to Defendants' Responsive Claim Construction Brief, Dkt. No. 210; *see also* Ex G at 8-9, attached to Defendants' Responsive Claim Construction Brief, Dkt. No. 210 (explaining that the "claimed virtual machine means is not just a JavaOS or a Java Virtual Machine" because it is comprised of the virtual function processor, the message instruction means, and the virtual message processor). Accordingly, the court rejects Defendants' argument that the "virtual machine means" must "process[] instructions in a hardware/operating system-independent language on the communication device."

In contrast to Defendants' proposed construction, the court finds that CardSoft's proposed construction of "virtual machine means" – i.e., "a computer programmed to emulate a hypothetical computer for applications relating to transport of data, including payment terminal devices in which message processing and communication comprise a significant proportion of the operation of the device" – is supported by the common specification of the patents-in-suit. For example, the specification states that "[a] virtual machine is [a] computer programmed to emulate a hypothetical computer." *See, e.g.*, '945 Patent at 3:40-41. However, although the specification states that "[t]he virtual machine therefore lends itself particularly to applications

Case: 14-1335 CASE PARTICIPANTS ONLY Document: 31 Page: 102 Filed: 07/10/2014

relating to communications, such as payment terminal devices and other devices in which message processing and communication comprise a significant proportion of the operation of the device," *see id.* at 4:51-65, this does not need to be a part of the court's construction. Accordingly, the court construes "virtual machine means" and "virtual machine" to mean "a computer programmed to emulate a hypothetical computer for applications relating to transport of data."

b. "emulatable in different computers having incompatible hardwares or operating systems"

Plaintiffs' Proposed Construction	**Defendants' Proposed Construction**
Capable of being implemented on computers having different hardware or operating systems.	The virtual machine means of the claimed communications device processes instructions expressed in a language that is hardware/operating system-independent so that the claimed virtual machine means can also be implemented, without compiling to a hardware/operating system-specific code or otherwise altering the virtual machine means or the instructions it processes, on other computers having hardware that is incompatible with that of the claimed device and on yet other computers having operating systems that are incompatible with that of the claimed device

Claim 1 of the '945 Patent, which is representative of the use of the phrase "emulatable in different computers having incompatible hardwares or operating systems," recites as follows:

> A communication device which is arranged to process messages for communications, comprising a virtual machine means which includes
>
> a virtual function processor and function processor instructions for controlling operation of the device, and
>
> message induction means including a set of descriptions of message data;
>
> a virtual message processor…,
>
> *wherein the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems.*

Id. at 50:49-67 (emphasis added). CardSoft urges the court to construe the phrase "emulatable in different computers having incompatible hardwares or operating systems" to mean "capable of being implemented on computers having different hardware or operating systems." Defendants, on the other hand, argue that the court should construe the phrase to mean "the virtual machine means of the claimed communications device processes instructions expressed in a language that is hardware/operating system-independent so that the claimed virtual machine means can also be implemented, without compiling to a hardware/operating system-specific code or otherwise altering the virtual machine means or the instructions it processes, on other computers having hardware that is incompatible with that of the claimed device and on yet other computers having operating systems that are incompatible with that of the claimed device." The parties' primary disputes are: (1) whether the virtual machine means must process instructions expressed in "a hardware/operating system-independent language;" and (2) whether the virtual machine must be implemented on various different computers "without compiling to a hardware/operating system-specific code or otherwise altering the virtual machine means or the instructions it processes." As discussed above, the court rejects Defendants' argument that the virtual machine must be expressed in "a hardware/operating system-independent language." Accordingly, in its analysis of this term, the court will address only Defendants' contention that the virtual machine cannot be compiled directly to the hardware-specific code of a particular processor.

Defendants argue that compiling to the hardware-specific code is outside the claim language because, if such compiling is done, then the virtual machine would be limited to operation on that one particular processor and would no longer be emulatable on a different, incompatible processor. Similarly, Defendants contend that programming the virtual machine in code that is specific to a particular operating system would limit operation of the virtual machine

to that single operating system and preclude its operation on a different, incompatible operating system. Defendants, therefore, urge the court to conclude that the "emulatable" limitation must be construed to recognize that it requires that the virtual machine not be compiled to a hardware/operating system-specific code.

As noted above, however, both Claim 5 and Claim 6 of the '945 Patent require that the virtual message processor and the virtual function processor, respectively, are implemented in the native code of the specific microprocessor in the device. As such, Defendants' proposed limitation is again at odds with the plain language of Claims 5 and 6 of the '945 Patent.[3] Furthermore, the common specification teaches that the "message processor 105 and protocol processor 106 are implemented *in native code of the payment terminal* and therefore operate at relatively high speed." '945 Patent at 10:26-29 (emphasis added). Thus, Defendants' proposed construction would also improperly read embodiments out of the scope of the patents-in-suit. As such, the court rejects Defendants' proposed construction.

Plaintiff's proposed construction,[4] however, is more consistent with the plain meaning of the words of the claim and with the common specification of the patents-in-suit. For example, the specification states that "[d]ifferent incompatible computers may be programmed to emulate the same hypothetical computer. Any computer programmed to emulate the hypothetical computer will thus be capable of executing programs for the virtual computer." *See, e.g.*, '945 Patent 3: 40-46. The specification further states that "[t]he virtual machine 101, 102, 103 can be

3 Defendants again rely on prosecution history statements discussed in the court's analysis of the "virtual machine means." In accordance with the court's previous analysis, the court rejects Defendants' contention that any of the prosecution history statements on which they rely constitute a clear disclaimer of virtual machines that have been compiled down to the hardware-specific code of the processor.

4 "Capable of being implemented on computers having different hardware or operating systems."

adapted for many different hardware 100 arrangements (i.e. many different brands of payment terminal). Different arrangements of hardware 100 can therefore be controlled by the same application software 104." *See id.* at 10:2-7. Thus, the court construes the phrase "emulatable in different computers having incompatible hardware or operating systems" to mean "capable of executing programs on different computers having incompatible hardware or operating systems." *See id.* at 3:43-46 ("Any computer programmed to emulate the hypothetical computer will thus be *capable of executing programs* for the virtual computer.") (emphasis added)).

c. "virtual message processor"

Plaintiffs' Proposed Construction	Defendants' Proposed Construction
A program module which processes messages, including assembling, disassembling and/or comparing messages, for communication to and/or from a payment terminal device.	Software that emulates a physical processor on the claimed communications device to handle the claimed messages in accordance with instructions expressed on the communications device in a hardware/operating system-independent language.

The parties' only dispute regarding the claimed "virtual message processor" is whether the processor must "handle the claimed messages in accordance with instructions expressed on the communications device in a hardware/operating system-independent language." With regard to this term, Defendants argue that their proposed limitation is required by the following description of the "virtual message processor":

> The message processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The message processor instructions are preferably virtual instructions to be expressed only in the language defined by the message processor means- and thus never requiring translation to any real hardware processor.

'945 Patent at 4:5-11. Furthermore, Defendants contend that the prosecution history confirms that their proposed limitation is necessary. In particular, Defendants argue that when the

applicant amended Claim 1 of the ’945 Patent to add the “message instruction means” to the “virtual message processor” limitation, the applicant argued:

> As stated in the Specification page 7, providing a separate virtual message processor allows for ‘faster, simpler programming.’ Stern does not teach the provision of the claimed virtual machine with a dedicated virtual message processor. That is, if a Java Virtual Machine as described in Stern is used to perform messaging, each application developed would be required to adjust to the characteristics of the different devices that the application was to execute on, such as screen width and fonts.
>
> The claimed virtual message processor removes this burden from the development of the application and places it on the software platform that resides on the device. This relieves the application developers of the burden of programming to the physical characteristics of the platform that application will execute on.

Ex. G at 13-14, attached to Defendants’ Responsive Claim Construction Brief, Dkt. No. 210; *see also id*., Ex. D at 2-3.

The specification explains that the “virtual machine processor” includes a “message processor 105” that is “implemented in software code.” ’945 Patent at 10:18-20. The specification then explicitly states that the “message processor 105 … [is] *implemented in the native code* of the payment terminal and therefore operates at relatively high speed.” *Id*. at 10:26-29. When read in light of the specification, the claimed “virtual message processor” is implemented in the native code of the communications device. The court disagrees with Plaintiffs that the doctrine of claim differentiation requires the court to hold otherwise. Although claim 5 requires that “the message processor be implemented in the native software code of the microprocessor,” claim differentiation does not trump the clear import of the specification. *See Edward Lifesciences LLV v. Cook Inc*., 582 F.3d 1322, 1332 (Fed. Cir. 2009) (“claim differentiation is a rule of thumb that does not trump the clear import of the specification.”). Here, the specification makes clear that the claimed “virtual message processor” is implemented

Case 2:13-cv-00290-JRG Document 261 Filed 09/21/2018 Page 19 of 30 PageID #: 7828

in the native code of the communications device. The specification, however, states that the claimed invention is not limited to devices configured to process payment transactions. *See id.* at 3:50-55. The court, therefore, rejects CardSoft's proposed "payment terminal device" limitation.

In conclusion, the court construes "virtual message processor" to mean "software implemented in the native code of the communications device that processes messages, including assembling, disassembling and/or comparing messages, for communication to and/or from a communications device."

d. "virtual function processor"

Plaintiffs' Proposed Construction	Defendants' Proposed Construction
A program module which controls and/or selects general operations of a payment terminal device.	Software that emulates a physical processor on the claimed communications device to control the operation of the device, and that interfaces with an application running on the device to process instructions from the application that are expressed on the communications device in a hardware/operating system-independent language.

Defendants again attempt to import a limitation, requiring that the "virtual function processor" "interface[] with an application running on the device to process instructions from the application that are expressed on the communications device in a hardware/operating system-independent language." Defendants' proposed limitation runs contrary to the language of Claim 6 of the '945 Patent, which requires that "the function processor is implemented in the native code of the microprocessor." Considering this, the court rejects Defendants' proposed construction.

In contrast to Defendants' proposed construction, CardSoft's proposed construction is supported by the common specification of the patents-in-suit. In particular, the common specification states that the claimed virtual machine includes "a function processor 107 the

operation of which is to control and select general operations of the device not specially controlled by the message and protocol processors 105, 106." '945 Patent at 10:34-37; *see also id.* at 5:15-18. The court, however, again notes that the claimed invention is not limited to "payment terminal" devices. *See id.* at 3:50-55. The court, therefore, construes "virtual function processor" to mean "software which controls and/or selects general operations of a communication device."

e. "message instruction means"

Plaintiffs' Proposed Construction	**Defendants' Proposed Construction**
Instructions arranged to provide directions for operation of a message processor, which include a description of a field of message data.	Governed by § 112, ¶ 6. **Function:** Using the hardware/operating system-independent language of the virtual machine means to specify operations that the virtual message processor carries out on the claimed messages. **Structure:** A set of instructions for processing the claimed messages, issued by the application and written and loaded onto the claimed communications device in a hardware/operating system-independent language.

Claim 1 of the '945 Patent, which is representative of the patents' use of the term "message instruction means," recites as follows:

> A communication device which is arranged to process messages for communications, comprising a virtual machine means which includes
>
> a virtual function processor and function processor instructions for controlling operation of the device, and
>
> *message induction means* [sic] including a set of descriptions of message data;
>
> a virtual message processor, which is arranged to be called by the function processor and which is arranged to carry out the message handling tasks of assembling the messages, disassembling messages and comparing the messages under the direction of the *message instruction means that is arranged to provide directions for operation of the virtual message processor,*

> whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task,
>
> wherein the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems.

Id. at 50:49-67 (emphasis added). The parties' dispute concerning the claimed "message instruction means" is two-fold: (1) whether the term is governed by 35 U.S.C. § 112, ¶ 6; and (2) whether the claimed message instructions must be "in a hardware/operating system-independent language."

First, Defendants contend that the term "message instruction means" is subject to means-plus-function treatment. It is well settled the use of the word "means" in a claim limitation raises a rebuttable presumption that the limitation is a means-plus-function limitation under § 112, ¶ 6. *Altiris, Inc. v. Symantec Corp.*, 318 F.3d 1363, 1375 (Fed. Cir. 2003). This presumption may be rebutted only if the patentee can demonstrate that the claim language itself recites sufficient structure to perform the claim function in its entirety. *Id.* Because the "message instruction means" limitation uses the word "means," the presumption that this limitation is a means-plus-function limitation applies. The recited function of the "message instruction means" is clear from the plain language of the claims – that is, "[providing] directions for operation of the virtual message processor." CardSoft argues that the independent claims of the patents-in-suit recite sufficient structure to perform this function in its entirety. The court, however, is not persuaded that CardSoft has overcome the presumption that is invoked by the use of the term "means." As such, the court rejects CardSoft's argument that the term "message instruction means" is exempt from means-plus-function treatment.

Defendants argue that the function of the "message instruction means" is "using the hardware/operating system-independent language of the virtual machine means to specify

operations that the virtual message processor carries out on the claimed messages." Defendants, however, offer no support for their proposed alteration of the function recited in the claims. Furthermore, Defendants' proposed construction attempts to import a limitation as to the "way" in which the function is performed. Federal Circuit precedent, however, makes clear that the "court must not import unclaimed functions into means-plus-functions limitations." *Applied Med. Res. Corp. v. U.S. Surgical Corp.*, 312 Fed. Appx. 326, 332 (Fed. Cir. 2009) (citing *JVW Enters., Inc. v. Interact Accessories, Inc.*, 424 F.3d 1324, 1331 (Fed. Cir. 2005). As such, the court rejects Defendants' proposed function and concludes that the function of the claimed "message instruction means" is "providing directions for operation of the virtual message processor." *See Lockheed Martin Corp. v. Space Sys./Loral, Inc.*, 324 F.3d 1308, 1319 (Fed. Cir. 2003) ("The function is properly identified as the language after the 'means for' clause and before the 'whereby' clause, because a whereby clause that merely states the result of the limitations in the claim adds nothing to the substance of the claim.").

With regard to the structure corresponding to this function, Defendants argue that the corresponding structure is "a set of instructions for processing the claimed messages, issued by the application and written and loaded onto the claimed communications device in a hardware/operating system-independent language." As with their other proposed constructions, Defendants again seek to import a limitation, requiring that the claimed message instructions be "in a hardware/operating system-independent language." Defendants' proposed construction, however, again runs afoul of the language of the claims. In particular, Claim 7 of the '945 Patent recites that "the message instruction means do not require translation to the native software code of the microprocessor." According to the doctrine of claim differentiation, this creates a presumption that Claim 1 (from which Claim 7 depends) must cover both "message instruction

means" that do not require translation to the native software code of the microprocessor and those that do require translation. *See Seachange Intern., Inc. v. C-COR, Inc.*, 413 F.3d 1361, 1368-69 (Fed. Cir. 2005). The court is not convinced that Defendants have overcome this presumption. Furthermore, Defendants' reliance on the specification for their proposed limitation is misplaced. Although the specification states that the "message processor instructions are preferably virtual instructions to be expressed only in the language defined by the message processor means- and thus never requiring translation to any real hardware processor," this is merely a embodiment of the claimed "message processor instructions." It is improper for the court to read such an embodiment into the claims. In summary, the court rejects Defendants' proposed structure because it is not supported by the claim language, common specification, or prosecution history of the patents-in-suit.

Having carefully reviewed the patents-in-suit, the court concludes that the structures corresponding to the function of "providing directions for operation of the virtual message processor" are: 13:29-14:2; 15:23-34; Figure 11 and Figure 8. The specification states that "FIG. 11 is a schematic diagram illustrating the structure of the message instruction means 109." '945 Patent at 13:29-30. It then goes on to explain that structure in detail. *Id.* at 13:30-14:2. Furthermore, the specification states that:

> the present invention includes another class of message instruction means, known as a "Form". Instead of a Data Representation as a message descriptor, a Form includes description of a Location of the data field in the Form. FIG. 8 is a display provided by a development tool enabling the programmer to prepare message instructions for a Form message.

Id. at 15:23-29. The specification also explains the structure of the "form" embodiment of the "message instruction means." *Id.* at 15:23-34. These are the only two structures identified in the specification that are clearly linked to the function of the "message instruction means."

In conclusion, the court construes the term “message instruction means” as follows: (1) the function is “providing directions for operation of the virtual message processor;” and (2) the structure is “13:29-14:2; 15:23-34; Figure 11 and Figure 8, and equivalents thereof.”

f. “function processor instructions” (’945 Patent: 1, 12, 14; ’683 Patent: 1)

Plaintiffs’ Proposed Construction	Defendants’ Proposed Construction
Instructions arranged to provide directions for operation of a function processor.	A set of instructions that control operation of the claimed communications device, written and loaded onto the communications device in the hardware/operating system-independent language of the virtual function processor.

The parties’ proposed constructions for the claim term “function processor instructions” differ in two material respects. First, Defendants’ proposed construction requires that the “function processor instructions” control the operation of the claimed communications, and second, Defendants’ proposed construction requires that the “function processor instructions” be written in the hardware/operating system-independent language. As to the first point, CardSoft does not dispute that the “function processor instructions” control the operation of the claimed communications device. Indeed, the claims expressly recite “function processor instructions for controlling operation of the device,” and the specification explains that the “function processor instructions” “control[] operation of the device.” ’945 Patent at 3:60-61; 7:26-27; 7:47. As such, the court agrees with Defendants that the “function processor instructions” is a set of instructions that control operation of the communications device.

With respect the parties’ second dispute, the court rejects Defendants’ contention that the “function processor instructions” must be written in a hardware/operating system-independent language. Defendants’ proposed limitation again runs contrary to the language of the claims. Specifically, Claim 8 of the ’945 Patent recites “wherein the function processor instruction

means are implemented in software defined by the function processor means and do not require translation to the native code of the microprocessor." As discussed above, this claim creates a presumption that because Claim 8 limits the function processor instruction means to implementation in software defined by the function processor, Claim 1 is not so limited and is broad enough to cover both function processor instructions implemented in software defined by the function processor and function processor instructions not implemented in software defined by the function processor. Furthermore, Defendants' reliance on statements in the specification indicating that the function processor instructions "preferably" never require translation to any real hardware processor do not overcome this presumption. '945 Patent at 5:19-25. These statements merely describe an embodiment of the invention claimed by the patents-in-suit and such embodiments cannot be read into the claims.

In conclusion, the court construes "function processor instructions" to mean "a set of instructions that control operation of the communications device."

V. CONCLUSION

The court adopts the constructions set forth in this opinion for the disputed terms of the patents-in-suit. The parties are ordered that they may not refer, directly or indirectly, to each other's claim construction positions in the presence of the jury. Likewise, the parties are ordered to refrain from mentioning any portion of this opinion, other than the actual definitions adopted by the court, in the presence of the jury. Any reference to claim construction proceedings is limited to informing the jury of the definitions adopted by the court.

It is so ORDERED.

SIGNED this 23rd day of September, 2011.

Charles Everingham IV

CHARLES EVERINGHAM IV
UNITED STATES MAGISTRATE JUDGE

Case: 14-1135 CASE PARTICIPANTS ONLY Document: 31 Page: 115 Filed: 02/18/2014

Memorandum Order, Dated May 28, 2012

**IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

CARDSOFT, INC., et al.	§	
	§	
v.	§	Case No. 2:08-CV-98-RSP
	§	
VERIFONE SYSTEMS, INC., et al.	§	

MEMORANDUM ORDER

Before the Court is VeriFone's Motion for Leave to Supplement Invalidity Contentions (Dkt. No. 230, filed August 8, 2011), Hypercom's Notice of Joinder in VeriFone's motion (Dkt. No. 234, filed August 19, 2011), and CardSoft's Motion to Strike the invalidity contentions (Dkt. No. 232, filed August 18, 2011). Because VeriFone has not shown good cause exists for amending its invalidity contentions, VeriFone's motion is **DENIED**, and CardSoft's motion is **GRANTED**.

APPLICABLE LAW

A party's invalidity contentions are deemed to be the party's final invalidity contentions unless amendment or supplementation is permitted by the Local Patent Rules. P.R. 3-6. In limited circumstances, amendment of invalidity contentions is permitted as of right. P.R. 3-6(a). Otherwise, amendment "may be made only by order of the Court, which shall be entered only upon a showing of good cause." P.R. 3-6(b). The Court considers four factors to determine whether good cause has been shown: (1) the explanation for the party's failure to timely move for leave to amend, (2) the importance of the amendment, (3) potential prejudice from allowing the amendment, and (4) the availability of a continuance to cure such prejudice. *See S&W Enterprises, L.L.C. v. SouthTrust Bank of Alabama, NA*, 315 F.3d 533, 536 (5th Cir. 2003).

DISCUSSION

VeriFone seeks leave to amend its invalidity contentions to include two additional prior art references: 1) the Omni 300 prior art and 2) the Open Terminal Architecture (OTA) prior art.

VeriFone's invalidity contentions were due in 2009. VeriFone explains that CardSoft amended its infringement contentions on August 12, 2010 to include the Omni 3200, Omni 3200 SE, Omni 3210, and Omni 3210 SE payment terminals. Dkt. No. 230 at 3. At that point, VeriFone's counsel began to review all 251 pages of the infringement contentions and all the cited documents, requiring "many hours which was spread out over several months." *Id.* In the course of this review, counsel determined that these payment terminals used the TXO software, which was different from the software used by the previously accused payment terminals. *Id.* In April 2011, VeriFone found the Omni 300 Programmers Manual Guide from 1994, which discloses the TXO software used by the Omni 300 payment terminal. *Id.* Although the Omni 300 is not accused of infringement, VeriFone maintains that this same software is used by the newly accused Omni 3200 payment terminals. *Id.* In April and May 2011, VeriFone's counsel searched for "prior art disclosing the Open Terminal Architecture and payment terminals programmed with the Open Terminal Architecture," and discovered the Open Terminal Architecture Specification. *Id.* at 4. VeriFone waited until May 2011 to produce the references that it found. *Id.* VeriFone waited until August 5, 2011 to serve amended invalidity contentions based upon these references. *Id.* at 5.

CardSoft argues that its original infringement contentions, which were served on June 15, 2009, provided ample notice to enable VeriFone to discover these prior art references. Dkt. No. 242 at 3. Although the Omni 3200, Omni 3200 SE, Omni 3210, and Omni 3210 SE payment terminals were not accused at that time, the infringement contentions did accuse "the Omni 3000 family of terminals (including Omni 3350 and Omni 3750 and any past, present or future

variants and generations" of infringement. *Id.* CardSoft notes that as of April 2011, the Markman hearing was set for July 20, 2011, and was eventually held on August 8, 2011. *Id.*

The Court is not satisfied that VeriFone acted diligently to discover the Omni 300 and OTA prior art. These references are related to VeriFone's own products, which it should have uncovered even if CardSoft never accused the Omni 3200 payment terminals of infringement. Furthermore, it took VeriFone nearly eight months to produce the references, and then another four months to disclose its invalidity contentions based upon the references. The Court is not persuaded that this shows diligence on VeriFone's part. VeriFone's lack of an adequate explanation to explain its delay weighs strongly against granting leave.

The importance of these references are quite clear, which weighs in favor of granting VeriFone leave to amend. However, their importance only adds to the prejudice suffered by CardSoft because it was deprived of a meaningful opportunity to consider these references during the claim construction process. Because claim construction was nearly over by the time VeriFone disclosed its invalidity contentions based on the Omni 300 and OTA prior art, and claim construction is now complete, a continuance is not available to cure CardSoft's prejudice. The Court concludes that VeriFone has not shown that good cause exists to grant VeriFone leave to amend its invalidity contentions. Accordingly, the Court **DENIES** VeriFone's Motion for Leave to Amend its Invalidity Contentions (Dkt. No. 230), and **GRANTS** CardSoft's Motion to Strike VeriFone and Hypercom's Invalidity Contentions (Dkt. No. 232).

SIGNED this 28th day of May, 2012.

Roy S. Payne

ROY S. PAYNE
UNITED STATES MAGISTRATE JUDGE

Memorandum Order, Dated June 3, 2012

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

CARDSOFT, INC., et al.	§	
	§	
v.	§	Case No. 2:08-CV-98-RSP
	§	
VERIFONE SYSTEMS, INC., et al.	§	

MEMORANDUM ORDER

Before the Court is Plaintiff CardSoft Inc.'s Motion to Exclude Expert Reports, Declarations, and Testimony of Scott M. Nettles (Dkt. No. 284, filed February 17, 2012). The Court finds that the motion should be **GRANTED-IN-PART** and **DENIED-IN-PART**.

APPLICABLE LAW

An expert witness may provide opinion testimony if "(a) the expert's scientific, technical, or other specialized knowledge will help the trier of fact to understand the evidence or to determine a fact in issue; (b) the testimony is based on sufficient facts or data; (c) the testimony is the product of reliable principles and methods; and (d) the expert has reliably applied the principles and methods to the facts of the case." Fed. R. Evid. 702. A trial court is "charged with a 'gatekeeping role,' the objective of which is to ensure that expert testimony admitted into evidence is both reliable and relevant." *Sundance, Inc. v. DeMonte Fabricating Ltd.*, 550 F.3d 1356, 1360 (Fed. Cir. 2008).

DISCUSSION

Defendants VeriFone, Hypercom, and Ingenico intend to call Dr. Scott M. Nettles as an expert witness at trial. Dr. Nettles will offer his opinion that Defendants do not infringe the patents asserted by CardSoft. CardSoft raises several objections to Dr. Nettles' opinions.

A. Opinion Regarding "Virtual Machine" / "Virtual Machine Means"

CardSoft objects to Dr. Nettles opinion that the word "emulate" in the Court's construction of the terms "virtual machine" or "virtual machine means" requires that the virtual machine means translate instructions into the appropriate native code. Dkt. No. 284 at 4-6. CardSoft points to paragraphs 52 and 64 of Dr. Nettles' report as examplars:

> 52. Mr. Cole's report, in the portion describing the invention, also confirms and supports this understanding. For example, in paragraph 34 he says "As illustrated in Figure 2 of the patent, the virtual processor was an element within a conceptually layered structure that permitted application programs to function on multiple hardware platforms" and Mr. Cole includes Figure 2 as background for his analysis. The important point here is that he confirms that the virtual processor was an element of the layer structure and not the entire structure itself. In paragraph 36 he says: "The virtual machine, in conjunction with the hardware application [sic, abstraction] layer software and hardware driver software, then translated the application program instructions to instructions appropriate to the specific computer hardware," making it clear he understand the HW Driver and HAL to be separate and distinct from the virtual processors. Mr. Cole also made clear that he understands that a virtual machine must translate instructions into the appropriate instructions for the specific computer hardware. This translating is how a virtual machine emulates a hypothetical computer and in so translating/emulating provides the instructions to the actual computer hardware.
>
> * * *
>
> 64. None of the accused systems have any such virtual machine that translates application program instructions and thus none of the accused systems are emulating a hypothetical computer. The remainder of Mr. Cole's report, however, assumes a different meaning of "virtual machine" that contradicts the common industry understanding, the patents, and the Court's Claim Construction ruling, and nowhere for any accused product does he show evidence of any translation of application program instructions. Mr. Cole could not point to such virtual machine on any terminal because all application program instructions are first compiled to the native code of the microprocessor on the terminal and execute directly on the microprocessor without any translating being necessary or possible. Despite his inability to point to such a translation, he nevertheless provides a conclusion that accused systems are "emulating a hypothetical computer" without the facts necessary to back up that conclusion.

Nettles Rebuttal Rep. ¶¶ 52 and 64.

Defendants argue that Dr. Nettles is not offering an opinion that contradicts the Court's claim construction, but is merely rebutting an opinion offered by J. Tipton Cole, CardSoft's expert. Dkt. No. 305 at 6-8. Furthermore, Defendants contend that Dr. Nettles is merely characterizing the Court's construction. *Id.* at 8.

The Court is concerned that the cited testimony may confuse the jury as to the proper construction of the term "virtual machine." In particular, Dr. Nettles' opinion that "Mr. Cole could not point to such virtual machine on any terminal because all application program instructions are first compiled to the native code of the microprocessor on the terminal and execute directly on the microprocessor without any translating being necessary or possible" is similar to an argument raised during claim construction. In rejecting Defendants' proposed construction for the term "virtual machine," the court noted that the message processor and the function processor can be implemented in the native software code of the processor, and therefore translation would not be necessary for the application to execute on the microprocessor. *See* Claim Construction Order, Dkt. No. 251 at 12. Accordingly, testimony that the virtual machine must translate instructions into native code for execution by the processor will be excluded. CardSoft's objection is **SUSTAINED**.

B. Opinion Regarding "Emulatable in Different Computers Having Incompatible Hardware or Operating System"

CardSoft objects to Dr. Nettles' opinion, allegedly offered at his deposition, that the virtual machine means "must be capable of executing programs on 'both' different hardware and different operating systems." Dkt. No. 284 at 6. CardSoft contends that this opinion runs afoul of the Court's construction of the term "emulatable in different computers having incompatible hardware or operating systems." *Id.*

The Court has reviewed the cited deposition testimony, and concludes that the deposition testimony is too ambiguous for the Court to rule on this particular objection. The objection is **OVERRULED** at this time; however CardSoft is free to make an objection at trial.

C. Opinion Regarding "Virtual Function Processor" and "Virtual Message Processor"

CardSoft objects to Dr. Nettles' opinion that the claims require the virtual function processor and the virtual message processor be distinct, and because both limitations are allegedly met by the operating system of the accused devices, there can be no infringement:

> 51. Based on Figure 2 and the discussion in Col 9:55-10:49, we can understand the layers of the system and how they interrelate. The lowest layer 100 is the hardware. Above that is the Virtual Machine (VM), which is composed of three layers, from bottom to top: 101 HW Drivers, 102 HAL, and 103 VM Processors. Above VM Processor layer is the Application layer 104 followed by the Data 112. Several observations are important to make about this structure. First, the Hardware 100, Application 104, and Data 112 layers are separate and distinct from the Virtual machine. Thus, parts of the accused systems that are found in the hardware, the applications, or the data cannot meet any limitations that require a component to be part of the Virtual Machine. Second, within the Virtual Machine 101 102 103, the HW Drivers 101, HAL 102, and VM Processors 103 are separate and distinct from each other and, as noted above, the HW Drivers 101 and HAL 102 are known in the prior art. Also, the HW Drivers 101 include existing BIOSes, OSes, and drivers, while the VM Processors 103 include the Function processor 107 and the Message Processor 105. There are several implications for infringement because of this. *First, because meeting all the limitations require the existence of function and message processors and these are separate and distinct from the OS, accusations of the OS alone cannot lead to all the limitations being met. Further, accusing portions of the OS of being the Function or Message processor also will fail to establish infringement because of the requirement of distinctness.*

Nettles Rebuttal Rep. ¶ 51 (emphasis added). CardSoft contends that Dr. Nettles' "requirement of distinctness" is not supported by the Court's claim construction, and is not proper under the applicable law. Dkt. No. 284 at 8-9.

Defendants respond that CardSoft "confuses the Court's construction of these terms with Dr. Nettles' expert non-infringement analysis applying the Court's construction of these terms." Dkt. No. 305 at 1. Defendants argue that the requirement of distinctness is not a misconception of the Court's claim construction, but it instead factors into the non-infringement analysis. *Id.* In response to CardSoft's allegation that "Dr. Nettles was 'unable to provide any legal basis for his newly-minted "requirement of distinctness"' at his deposition," Defendants cite Dr. Nettles' explanation for the requirement:

> A. *Well, my understanding of the claim is that the -- so if I look at the claim, I see that it calls out a virtual function processor and a virtual message processor and an operating system as specific disparate components*. And, furthermore, the virtual function processor and the virtual message processor are components of the virtual machine means, and that virtual machine means has to be capable of executing programs on different computers having incompatible hardware or operating systems. And so both because the claims call them out as distinct components, and because I can't really imagine a way that this virtual machine means would be emulateable on different operating systems, if it was the operating system, I think that these things have to be distinct.

Dkt No. 305 at 13-14 citing Nettles Depo. at 77:1-16 (emphasis added).

The Court disagrees with Defendants' characterization of Dr. Nettles' opinion. Dr. Nettles' "requirement of distinctness" does not merely arise from the application of the claims to the accused subject matter, but is instead an opinion on the proper scope of the claims. This is clear from the very first sentence of Dr. Nettles' testimony where he explains that the requirement of distinctness comes from his "understanding" of the claims. Claim construction arguments will not be presented to the jury due to the high risk that claim construction arguments create confusion. *Cordis Corp. v. Boston Scientific Corp.*, 561 F.3d 1319, 1337 (Fed. Cir. 2009). Therefore, CardSoft's objection is **SUSTAINED**.

CONCLUSION

For the foregoing reasons, CardSoft's Motion to Exclude Expert Reports, Declarations, and Testimony of Scott M. Nettles is **GRANTED-IN-PART** and **DENIED-IN-PART**.

SIGNED this 3rd day of June, 2012.

ROY S. PAYNE
UNITED STATES MAGISTRATE JUDGE

Verdict Form, Dated June 8, 2012

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

CARDSOFT, INC. and CARDSOFT (ASSIGNMENT FOR THE BENEFIT OF CREDITORS), LLC	§	
v.	§	Case No. 2:08-CV-98-RSP
VERIFONE SYSTEMS, INC.; VERIFONE INC.; HYPERCOM CORP.; INGENICO S.A.; INGENICO CORP.; and INGENICO INC.	§	

VERDICT FORM

In answering these questions, you are to follow all of the instructions I have given you.

1. **Did CardSoft prove by a preponderance of the evidence that VeriFone, Hypercom, or Ingenico literally infringed the claims of the '945 and '683 patents listed below?**

Answer "Yes" or "No" for each patent claim. If you find the claim infringed, answer "Yes," otherwise, answer "No."

	VeriFone Infringed?	Hypercom Infringed?	Ingenico Infringed?
Claim 11 of the '945 Patent	yes	yes	NO
Claim 1 of the '683 Patent	yes	yes	NO

2. Did VeriFone, Hypercom, or Ingenico prove by clear and convincing evidence that any of the following claims are invalid in view of the prior art?

Answer "Yes" or "No" for each claim. If you find the claim invalid, answer "Yes," otherwise, answer "No."

	Invalid as Anticipated?
Claim 11 of the '945 Patent	NO
Claim 1 of the '683 Patent	NO

ANSWER QUESTIONS 3-7 ONLY IF YOU FOUND AT LEAST ONE CLAIM BOTH INFRINGED AND NOT INVALID

3. Did CardSoft prove by clear and convincing evidence that VeriFone willfully infringed the '945 or '683 Patents?

Answer "Yes" or "No."

NO

4. Did CardSoft prove by clear and convincing evidence that Hypercom willfully infringed the '945 or '683 Patents?

Answer "Yes" or "No."

NO

5. Did CardSoft prove by clear and convincing evidence that Ingenico willfully infringed the '945 or '683 Patents?

Answer "Yes" or "No."

NO

6. For each Defendant, what sum of money do you find that CardSoft proved by a preponderance of the evidence would fairly and reasonably compensate CardSoft for each Defendants' infringement of the claims that you found were infringed and not invalid?

For VeriFone: $13,148,958.00

For Hypercom: $2,245,863.00

For Ingenico: $0

7. If you awarded money damages in Question 6, what royalty rate, expressed either as a percentage or a per unit amount, did you apply in reaching that amount?

$3 per Unit

The jury foreperson should sign and date the Verdict Form and return it to the Bailiff.

June 8, 2012
Date

Order, Dated March 28, 2013

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

CARDSOFT, INC., et al.	§	
	§	
v.	§	Case No. 2:08-CV-98-RSP
	§	
VERIFONE HOLDINGS, INC., et al.	§	

ORDER

Before the Court are the parties' motions made under Federal Rule of Civil Procedure 50(a):

- Defendant Hypercom's Motion for Judgment as a Matter of Law of Non-Infringement (Dkt. No. 380);
- Defendant Verifone's Rule 50 Motion for Judgment as a Matter of Law of Non-Infringement (Dkt. No. 381);
- Defendant Ingenico's Rule 50 Motion for Judgment as a Matter of Law of Non-Infringement (Dkt. No. 382); and
- Plaintiff CardSoft's Rule 50(a) Motion for Judgment as a Matter of Law of No Invalidity (Dkt. No. 387).

To the extent that these motions were not granted prior to the submission of this case to the jury, these motions are **DENIED** without prejudice to the pending post-trial motions (such as those made pursuant to Federal Rule of Civil Procedure 50(b) and 59).

SIGNED this 28th day of March, 2013.

Roy S. Payne
ROY S. PAYNE
UNITED STATES MAGISTRATE JUDGE

Case: 14-1135 Document: 12 Page: 128 Filed: 02/18/2014

Memorandum Order, Dated September 30, 2013

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

CARDSOFT, INC., et al.	§	
	§	
v.	§	Case No. 2:08-CV-98-RSP
	§	
VERIFONE HOLDINGS, INC., et al.	§	

MEMORANDUM ORDER

On June 8, 2012, a jury returned a verdict finding that Defendants VeriFone Systems, Inc. and VeriFone, Inc. (collectively, "VeriFone") infringe claim 11 of U.S. Patent No. 6,934,945 and claim 1 of U.S. Patent No. 7,302,683. (Verdict, Dkt. No. 389.) The jury found that the asserted claims are not invalid as anticipated. (*Id.*) The jury awarded $13,148,958.00 in damages for VeriFone's infringement, and applied a running royalty rate of $3.00 per unit. (*Id.*) Before the Court are Defendant VeriFone Systems, Inc.'s and VeriFone, Inc.'s Renewed Motion for Judgment as a Matter of Law (hereinafter "JMOL Mot.", Dkt. No. 425) and Corrected Motion for New Trial (hereinafter "NT Mot.", Dkt. No. 432).

APPLICABLE LAW

A. Judgment as a Matter of Law

Judgment as a matter of law is appropriate "[i]f a party has been fully heard on an issue during a jury trial and the court finds that a reasonable jury would not have a legally sufficient evidentiary basis to find for the party on that issue . . . ." Fed. R. Civ. P. 50(a) & (b). "The grant or denial of a motion for judgment as a matter of law is a procedural issue not unique to patent law, reviewed under the law of the regional circuit in which the appeal from the district court would usually lie." *Finisar Corp. v. DirectTV Group, Inc.*, 523 F.3d 1323, 1332 (Fed. Cir. 2008).

Under Fifth Circuit law, a court is "especially deferential" to a jury's verdict, and will not reverse the jury's findings unless they are not supported by substantial evidence. *Baisden v. I'm Ready Productions, Inc.*, 693 F.3d 491, 499 (5th Cir. 2012). A motion for judgment as a matter of law must be denied "unless the facts and inferences point so strongly and overwhelmingly in the movant's favor that reasonable jurors could not reach a contrary conclusion." *Id.* at 498 (citation omitted). In other words, "[t]here must be more than a mere scintilla of evidence in the record to prevent judgment as a matter of law in favor of the movant." *Arismendez v. Nightingale Home Health Care, Inc.*, 493 F.3d 602, 606 (5th Cir. 2007).

The Court must review all evidence in the record, and draw all reasonable inferences in favor of the nonmoving party. *Reeves v. Sanderson Plumbing Prods., Inc.*, 530 U.S. 133, 150 (2000). However, "[c]redibility determinations, the weighing of the evidence, and the drawing of legitimate inferences from the facts are jury functions, not those of a judge." *Id.* "[T]he court should give credence to the evidence favoring the nonmovant as well as that 'evidence supporting the moving party that is uncontradicted and unimpeached, at least to the extent that that evidence comes from disinterested witnesses.'" *Id.* at 151 (citation omitted).

B. New Trial

After a jury trial, the Court may grant a new trial on all or some issues "for any reason for which a new trial has heretofore been granted in an action at law in federal court." Fed. R. Civ. P. 59(a). "A new trial may be granted, for example, if the district court finds the verdict is against the weight of the evidence, the damages awarded are excessive, the trial was unfair, or prejudicial error was committed in its course." *Smith v. Transworld Drilling Co.*, 773 F.2d 610, 612-13 (5th Cir. 1985). When a motion for a new trial questions the sufficiency of the evidence, the Court must review all the evidence "in a light most favorable to the jury's verdict" and the motion must be denied "unless the evidence points so strongly and overwhelmingly in favor of

one party that the court believes that reasonable persons could not arrive at a contrary conclusion." *Dawson v. Wal-Mart Stores, Inc.*, 972 F.2d 205, 208 (5th Cir. 1992).

DISCUSSION

A. Direct Infringement

To prove infringement, the patentee must show the presence of every element or its equivalent in the accused device. *Lemelson v. United States*, 752 F.2d 1538, 1551 (Fed. Cir. 1985). Determining infringement is a two-step process: "First, the claim must be properly construed, to determine the scope and meaning. Second, the claim, as properly construed must be compared to the accused device or process." *Absolute Software, Inc. v. Stealth Signal, Inc.*, 659 F.3d 1121, 1129 (Fed. Cir. 2011) (citing *Carroll Touch, Inc. v. Electro Mech. Sys., Inc.*, 15 F.3d 1573, 1576 (Fed. Cir. 1993)). To infringe a means-plus-function limitation, "the relevant structure in the accused device [must] perform the identical function recited in the claim and be identical or equivalent to the corresponding structure in the specification." *Odetics, Inc. v. Storage Tech. Corp.*, 185 F3d 1259, 1267 (Fed. Cir. 1999). "A determination of infringement is a question of fact that is reviewed for substantial evidence when tried to a jury." *ACCO Brands, Inc. v. ABA Locks Mfr. Co.*, 501 F.3d 1307, 1311 (Fed. Cir. 2007).

1. "Virtual Function Processor" and "Function Processor Instructions"

VeriFone contends that CardSoft failed to present substantial evidence that the Verix or Verix V terminals meet the "virtual function processor" and "function processor instructions" limitations. (JMOL Mot. at 5-8.) VeriFone also contends that the jury's finding that the limitations are present was against the great weight of the evidence and that a new trial is warranted. (NT Mot. at 6-7.) Both limitations are found in the two asserted claims. The Court found that "virtual function processor" means "software which controls and/or selects general operations of a communication device." (Claim Construction Order at 20, Dkt. No. 251.)

"Function processor instructions" means "a set of instructions that control operation of the communications device." (*Id.* at 25.)

At trial, CardSoft presented Joe Tipton Cole as its technical expert. As VeriFone concedes, Mr. Cole testified that the "virtual function processor" limitation is met in the Verix and Verix V terminals by the "main task" described in PTX 155 (the Verix V Operating System Programmers Manual), and by the "application idle engine" described in PTX 154 (the Verix/Verix V ACT2000 Programmers Manual). (JMOL Mot. at 5.) Therefore, there was substantial evidence to support the jury's finding that the "virtual function processor" limitation is present in the Verix and Verix V terminals. Moreover, the jury's finding was not against the weight of the evidence.

CardSoft also presented evidence that the Verix and Verix V terminals have "function processor instructions." Mr. Cole testified that the *GO instruction is an example of the "function processor instructions," and that this instruction is used to start the "main task." (6/5 a.m. Tr. 54:25-55:19.) Mr. Cole explained that PTX 155 shows that the "main task" is able to call other additional tasks. (*Id.* at 55:18-56:5.) PTX 155 states that the additional tasks "perform[] a specialized function such as printing a receipts, handling device input and output, modem communications, or controlling the overall program flow . . . ." (PTX 155 at 159, Dkt. No. 443-5.) Mr. Cole also explained to the jury how the application processing flow chart shown in PTX 154 (Verix/Verix V ACT2000 Programmers Manual) is evidence of "function processor instructions." Mr. Cole explained that the flow chart shows that "virtual function processor" detects an event and then "kick[s] off" the "function processor instructions" to process the event. (6/5 a.m. Tr. 57:1-58:14.)

VeriFone argues Mr. Cole improperly "merged the function processor instructions limitation into the virtual function processor limitation." (JMOL Mot. at 5-7.) Although Mr. Cole discussed evidence concerning both limitations at the same time, the record shows that Mr. Cole pointed to different parts of the Verix and Verix V operating systems for each limitation. Therefore, the Court finds that Mr. Cole's direct infringement opinion was not improper. The Court finds that there is substantial evidence to support the jury's finding that the "function processor instructions" limitation is present in the Verix and Verix V terminals, and that this finding was not against the weight of the evidence.

2. "Virtual Message Processor"

In its renewed motion for judgment of a matter of law, VeriFone argues that CardSoft failed to offer substantial evidence to support the jury's finding that Verix and Verix V terminals have a "virtual message processor" because none of the "virtual message processors" identified by Mr. Cole "assembled, disassembled or compared messages for communications between the terminal and the bank." (JMOL Mot. at 8-9.) VeriFone raises the same issue in its motion for a new trial. (NT Mot. at 8-9.) CardSoft objects to this portion of the motion because it raises a new ground for judgment as a matter of law that was not present in VeriFone's pre-verdict motion for judgment as a matter of law. (JMOL Resp. at 11.)

"It is well-settled in [the Fifth Circuit] that a motion for judgment as a matter of law filed post verdict cannot assert a ground that was not included in the motion for judgment as a matter of law made at the close of the evidence." *Morante v. American General Financial Center*, 157 F.3d 1006, 1010 (5th Cir. 1998). "It would be a constitutionally impermissible re-examination of the jury's verdict for the district court to enter judgment n.o.v. on a ground not raised in the motion for a directed verdict." *Id.* (quoting *Sulmeyer v. Coca Cola Co.*, 515 F.2d 835, 846 n.17 (5th Cir. 1975)).

In its pre-verdict motion, VeriFone limited its motion to the alleged failure of CardSoft to show that the "virtual message processor" was called by a "virtual function processor":

> VeriFone's terminals do not have at least the following claim elements, and VeriFone limits this motion to these elements for the purpose of judicial efficiency without conceding the presence of other claim elements in VeriFone's terminals: . . .
>
> 5. no virtual message processor, specifically no virtual message processor which is arranged to be called by the function processor.
>
> * * *
>
> VeriFone is entitled to judgment of noninfringement as a matter of law, because plaintiffs have set forth no evidence that any of the components mentioned as a possible "virtual message processor" is "arranged to be called" by any of the components mentioned as a possible "virtual function processor."

(Pre-Verdict JMOL Mot. at 9 and 12, Dkt. No. 381.) The Court agrees with CardSoft that VeriFone's pre-verdict motion was not sufficient to put CardSoft on notice of the particular grounds that it is now urging.

Moreover, the original basis for judgment as a matter of law was not re-urged in VeriFone's renewed motion for judgment as a matter of law. Because VeriFone's renewed motion for judgment as a matter of law with respect to the "virtual message processor" relies solely on a new basis for judgment as a matter of law, it must be denied. With respect to VeriFone's motion for a new trial, the Court has considered the evidence presented with respect to "virtual message processor" limitation and concludes that the jury's finding is not against the weight of the evidence.

3. "Message Instruction Means"

VeriFone contends that CardSoft failed to present substantial evidence that VeriFone's terminals satisfy the "message instruction means" limitation on two separate grounds: (1) that CardSoft failed to show that a "message instruction means" in the VeriFone terminals provides

directions for operation of the virtual message processor; and (2) that CardSoft failed to show that the "message instruction means" in the VeriFone terminals is identical or equivalent to the structure disclosed in the patent. (JMOL Mot. at 10-12.) VeriFone raises the same issues in its motion for a new trial. (NT Mot. at 10-11.)

The "message instruction means" is a means-plus-function limitation that is subject to the provisions of 35 U.S.C. § 112, paragraph 6. The claim term has the function of "providing directions for operation of the virtual message processor." (Claim Construction Order at 24.) The claim term's corresponding structure is disclosed in the patent at "13:29-14:2; 15:23-34; Figure 11 and Figure 8, and equivalents thereof." (*Id.*) During the trial, Mr. Cole recited the Court's construction of the term "message instruction means," and explained that the "message instruction means" in the VeriFone terminals satisfies "the function of providing directions for operation of the virtual message processor . . . and it does this by employing a structure that looks like Figure 11 in the patent." (6/5 a.m. Tr. 59:16-60:1.) Mr. Cole then explained the data structures used to accomplish this function with reference to those described in PTX 154 (Verix/Verix V ACT2000 Programmers Manual). (*Id.* at 60:2-63:9.) In light of the foregoing evidence, the Court finds that there was substantial evidence to support the jury's finding that the "message instruction means" identified in the VeriFone terminals provides directions for the operation of the virtual message processor.

Turning to VeriFone's second basis for judgment as a matter of law (that the accused structure must be identical or equivalent to the disclosed structure), CardSoft contends that this basis was not argued in VeriFone's pre-verdict motion and is therefore not properly raised for the first time in VeriFone's renewed motion. (JMOL Resp. at 14.)

With respect to the "message instruction means" limitation, VeriFone's pre-verdict motion only raised the issue of whether the structure in the VeriFone terminals satisfied the required function:

> VeriFone's terminals do not have at least the following claim elements, and VeriFone limits this motion to these elements for the purpose of judicial efficiency without conceding the presence of other claim elements in VeriFone's terminals: . . .
>
> 4. no message instruction means, specifically no message instruction means having the function of providing directions for operation of the virtual message processor . . .
>
> * * *
>
> Wholly absent, however, from Mr. Cole's testimony is any identification of how any of these possible message instruction means provide directions for operation of the virtual message processor. Therefore, no evidence shows this claim element in VeriFone's terminals, and summary judgment is appropriate.

(Pre-Verdict JMOL Mot. at 9 and 11, Dkt. No. 381.) Because VeriFone's pre-verdict motion for judgment as a matter of law did not raise the issue of whether the accused "message instruction means" structure is identical or equivalent to that disclosed in the asserted patent and failed to put CardSoft on notice of this issue, VeriFone's motion for judgment as a matter of law on this ground must be denied.

The Court has considered both grounds that have been raised in VeriFone's motion for a new trial. In light of the foregoing evidence and the evidence cited by the parties, the Court concludes that the jury's finding that the "message instruction means" is present in VeriFone's terminals is not against the weight of the evidence. The Court is satisfied that the evidence shows that any differences between the structure show in Figure 11 of the patent and the structure found in VeriFone's terminals are insubstantial.

4. "Virtual Machine Means" and "Emulatable"

VeriFone moves for judgment as a matter of law and a new trial on several grounds relating to the sufficiency of the evidence for the "virtual machine means" and "emulatable" limitations. (JMOL Mot. at 13-19; NT Mot. at 12-15.) The asserted claims require a "virtual machine means," which is "a computer programmed to emulate a hypothetical computer for applications relating to transport of data." (Claim Construction Order at 14.) The asserted claims specify that "the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems." '945 Pat., cl. 1 and '683 Pat., cl. 1. The Court found that this limitation means that the "virtual machine means" is "capable of executing programs on different computers having incompatible hardware or operating systems." (Claim Construction Order at 17.)

First, VeriFone argues that CardSoft failed to present any evidence that VeriFone's terminals have a "virtual machine means," or that the "virtual machine means" satisfies the "emulatable" limitation. (JMOL Mot. at 13-14.) VeriFone argues that Mr. Cole skipped over the "virtual machine means" limitation and improperly merged the limitation with the "emulatable" limitation. (*Id.*)

Mr. Cole testified that the "emulatable" limitation is met by the VeriFone terminals because the Verix operating systems run on several different processors. (6/5 a.m. Tr. 67:12-70:14.) Although Mr. Cole offered in short form his opinion that the Verix and Verix V operating systems have a "virtual machine means," VeriFone did not cross-examine him on the basis for his opinion, and therefore the jury was free to accept or reject this opinion evidence. *See Symbol Techs., Inc. v. Opticon, Inc.*, 935 F.3d 1569, 1576 (Fed. Cir. 1991). The Court finds that there was substantial evidence in the record that the Verix and Verix V operating systems

have a "virtual machine means," and that the "virtual machines means" in both are "capable of executing programs on different computers having incompatible hardware or operating systems."

Second, VeriFone contends that CardSoft failed to present sufficient evidence that the "virtual machine means' is "capable of executing programs on different computers having incompatible hardware or operating systems." (JMOL Mot. at 15-16.) VeriFone argues that Mr. Cole failed to offer evidence "that any of VeriFone's operating systems, which he accuses as the virtual machine, executes programs" and that Mr. Cole "acknowledged that [the] application programs on VeriFone's terminals are all compiled to the native code of the microprocessor of each [terminal]." (*Id.*)

During cross-examination, Mr. Cole testified that it was his opinion that the "virtual machine means" execute the application programs. (6/5 a.m. Tr. 126:9-20.) Mr. Cole testified that it was not inconsistent for the application programs to be executed by the terminal's microprocessor and the "virtual machine means" because the claims are not limited to a "virtual machine means" that acts as an interpreter. (*Id.* at 126:21-127:6.) Moreover, CardSoft presented evidence (through the cross-examination of VeriFone's technical expert Dr. Scott Nettles) that an application program written for VeriFone's platform can be executed on VeriFone terminals having different microprocessors and operating systems by simply recompiling the source code to target a different device. (6/6 p.m. Tr. 156:14-158:25.) In other words, no modification to the source code for an application program is necessary in order for it to execute on incompatible devices. The Court finds that there is substantial evidence to support the jury's conclusion that the "virtual machine means" is "capable of executing programs on different computers having incompatible hardware or operating systems."

Lastly, VeriFone argues that there is insufficient evidence that the "emulatable" limitation is satisfied because Mr. Cole did not analyze whether the "virtual machine means" in the VeriFone terminals are capable of executing programs having an incompatible operating system. (JMOL Mot. at 17.) The Court finds that this argument has no merit in light of the Court's construction. The "emulatable" limitation only requires showing that the "virtual machine means" is capable of executing programs on different computers having 1) incompatible hardware or 2) incompatible operating systems. CardSoft presented evidence that the "virtual machine means" is capable of executing programs on different computers having incompatible hardware (e.g. different microprocessors). Therefore, there is substantial evidence to support finding infringement of the "emulatable" limitation. The Court also finds that the jury's conclusion that the "virtual machine means" and "emulatable" limitations are present in VeriFone's terminals is not against the weight of the evidence.

5. Verix eVo and Mx (Linux) Terminals

VeriFone contends that there is no substantial evidence of infringement by VeriFone terminals running the Verix eVo or Mx (Linux) operating systems. VeriFone argues that Mr. Cole failed to offer individualized testimony for each of these terminal operating systems, that Mr. Cole admitted to not having reviewed the source code for these terminal operating systems, and that his infringement opinions improperly rely on the testimony of VeriFone witness David Faoro that the Verix, Verix V, and Verix eVo operating systems are effectively the same program.

On direct examination, Mr. Cole testified that he understood the Mx operating system operated in the same way as the Verix operating systems (for which he reviewed source code), and that this opinion is based upon the testimony of VeriFone's fact witnesses including David Faoro. (6/5 a.m. Tr. 52:2-5; 113:1-7.) On cross-examination, Mr. Cole specifically testified that

it was his opinion that the Verix, Verix V, and Verix eVo operating systems "are effectively the same operating system for the purposes of [his infringement analysis]." (*Id.* at 132:18-24.) Mr. Cole stated that the evidence available to him showed that that "[Verix, Verix V, and Verix eVO] operating systems form a single platform that people can use." (*Id.* at 133:20-25.) With respect to the Mx operating system, Mr. Cole was specifically asked to explain why Mr. Faoro's deposition testimony (which was played for the jury) convinced him that Mx operating system operated in the same way, and Mr. Cole gave his explanation. (*Id.* at 137:2-13; 138:4-14.)

VeriFone's cross-examination clearly established for the jury that Mr. Cole's infringement opinion as to operating systems for which he did not review source code relied on fact witness testimony that was shown to the jury, and that the jury was free to consider the import of that evidence on its own. At no point did VeriFone's counsel point out any differences among the operating systems that VeriFone contends is material to infringement. In the instant motions, VeriFone does not show that there was any testimony offered at trial showing that there were any differences between the operating systems material to infringement in order to rebut Mr. Cole's opinions or the testimony offered by VeriFone's own fact witnesses. The Court finds that the record developed at trial contains substantial evidence to support the jury's finding that VeriFone terminals running the Verix, Verix V, Verix eVo, and Mx operating systems infringe in the same manner. The Court also concludes that the jury's finding was not against the weight of the evidence offered at trial.

CardSoft raises two other issues with respect to this issue. First, CardSoft argues that VeriFone waived its argument by failing to properly object to the verdict form, which only asked whether VeriFone terminals infringed the asserted claims. With respect to this argument, the Court notes that there was substantial evidence offered at trial that at least VeriFone terminals

running the Verix or Verix V operating systems directly infringe the asserted claims. Therefore, VeriFone's failure to object to the form of the verdict provides an alternative and independent basis for denying VeriFone's motion with respect to the Verix eVo and Mx terminals.

Cardsoft also argues that VeriFone failed to identify any non-infringement positions based upon differences in the source code among the accused operating systems during discovery, and therefore should be precluded from making this argument post-verdict. In light of the Court's disposition of this motion, the Court finds it unnecessary to reach this argument.

B. Exclusion of Certain Opinions of Dr. Nettles

VeriFone moves for a new trial on the basis that the Court improperly struck paragraphs 51, 52, and 64 of Dr. Scott Nettles' expert report (VeriFone's technical expert). (NT Mot. at 2-6.) The Court's order striking these portions of Dr. Nettles' report set forth in full the opinions that were stricken, as well as the Court's reasoning for excluding them. (*See* Mem. Order, Dkt. No. 372.) After considering the arguments of the parties, the Court is convinced that the exclusion of those opinions was proper, and that a new trial is not warranted.

CONCLUSION

For the foregoing reasons, VeriFone's Motion for Judgment as a Matter of Law (Dkt. No. 425) is **DENIED**, and VeriFone's Corrected Motion for New Trial (Dkt. No. 432) is **DENIED**.

SIGNED this 30th day of September, 2013.

Roy S. Payne

ROY S. PAYNE
UNITED STATES MAGISTRATE JUDGE

Case: 14-1135 Document: 1-2 Page: 148 Filed: 02/18/2014

Memorandum Order, Dated October 28, 2013

**IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

CARDSOFT, INC., et al.	§	
	§	
v.	§	Case No. 2:08-CV-98-RSP
	§	
VERIFONE HOLDINGS, INC., et al.	§	

MEMORANDUM ORDER

On June 8, 2012, a jury returned a verdict finding that Defendant Hypercom Corporation infringes claim 11 of U.S. Patent No. 6,934,945 and claim 1 of U.S. Patent No. 7,302,683. (Verdict, Dkt. No. 389.) The jury found that the asserted claims are not invalid as anticipated. (*Id.*) The jury awarded $2,245,863 in damages for Hypercom's infringement, and applied a running royalty rate of $3.00 per unit. (*Id.*) Before the Court are Hypercom's Renewed Motion for Judgment as a Matter of Law (hereinafter "JMOL Mot.", Dkt. No. 426) and Hypercom's Motion for New Trial (hereinafter "NT Mot.", Dkt. No. 427).[1]

APPLICABLE LAW

A. Judgment as a Matter of Law

Judgment as a matter of law is appropriate "[i]f a party has been fully heard on an issue during a jury trial and the court finds that a reasonable jury would not have a legally sufficient evidentiary basis to find for the party on that issue . . . ." Fed. R. Civ. P. 50(a) & (b). "The grant or denial of a motion for judgment as a matter of law is a procedural issue not unique to patent law, reviewed under the law of the regional circuit in which the appeal from the district court

[1] The Court observes that Hypercom's motions are very similar in content to VeriFone's post-trial motions. At times, Hypercom has failed to change references to VeriFone and VeriFone-related testimony to Hypercom and Hypercom-related testimony. Where possible, the Court has attempted to find the relevant portions of the record in order to evaluate the merits of Hypercom's arguments.

would usually lie." *Finisar Corp. v. DirectTV Group, Inc.*, 523 F.3d 1323, 1332 (Fed. Cir. 2008).

Under Fifth Circuit law, a court is "especially deferential" to a jury's verdict, and will not reverse the jury's findings unless they are not supported by substantial evidence. *Baisden v. I'm Ready Productions, Inc.*, 693 F.3d 491, 499 (5th Cir. 2012). A motion for judgment as a matter of law must be denied "unless the facts and inferences point so strongly and overwhelmingly in the movant's favor that reasonable jurors could not reach a contrary conclusion." *Id.* at 498 (citation omitted). In other words, "[t]here must be more than a mere scintilla of evidence in the record to prevent judgment as a matter of law in favor of the movant." *Arismendez v. Nightingale Home Health Care, Inc.*, 493 F.3d 602, 606 (5th Cir. 2007).

The Court must review all evidence in the record, and draw all reasonable inferences in favor of the nonmoving party. *Reeves v. Sanderson Plumbing Prods., Inc.*, 530 U.S. 133, 150 (2000). However, "[c]redibility determinations, the weighing of the evidence, and the drawing of legitimate inferences from the facts are jury functions, not those of a judge." *Id.* "[T]he court should give credence to the evidence favoring the nonmovant as well as that 'evidence supporting the moving party that is uncontradicted and unimpeached, at least to the extent that that evidence comes from disinterested witnesses.'" *Id.* at 151 (citation omitted).

B. New Trial

After a jury trial, the Court may grant a new trial on all or some issues "for any reason for which a new trial has heretofore been granted in an action at law in federal court." Fed. R. Civ. P. 59(a). "A new trial may be granted, for example, if the district court finds the verdict is against the weight of the evidence, the damages awarded are excessive, the trial was unfair, or prejudicial error was committed in its course." *Smith v. Transworld Drilling Co.*, 773 F.2d 610, 612-13 (5th Cir. 1985). When a motion for a new trial questions the sufficiency of the evidence,

the Court must review all the evidence "in a light most favorable to the jury's verdict" and the motion must be denied "unless the evidence points so strongly and overwhelmingly in favor of one party that the court believes that reasonable persons could not arrive at a contrary conclusion." *Dawson v. Wal-Mart Stores, Inc.*, 972 F.2d 205, 208 (5th Cir. 1992).

DISCUSSION

A. Direct Infringement

To prove infringement, the patentee must show the presence of every element or its equivalent in the accused device. *Lemelson v. United States*, 752 F.2d 1538, 1551 (Fed. Cir. 1985). Determining infringement is a two-step process: "[f]irst, the claim must be properly construed, to determine the scope and meaning. Second, the claim, as properly construed must be compared to the accused device or process." *Absolute Software, Inc. v. Stealth Signal, Inc.*, 659 F.3d 1121, 1129 (Fed. Cir. 2011) (citing *Carroll Touch, Inc. v. Electro Mech. Sys., Inc.*, 15 F.3d 1573, 1576 (Fed. Cir. 1993)). To infringe a means-plus-function limitation, "the relevant structure in the accused device [must] perform the identical function recited in the claim and be identical or equivalent to the corresponding structure in the specification." *Odetics, Inc. v. Storage Tech. Corp.*, 185 F3d 1259, 1267 (Fed. Cir. 1999). "A determination of infringement is a question of fact that is reviewed for substantial evidence when tried to a jury." *ACCO Brands, Inc. v. ABA Locks Mfr. Co.*, 501 F.3d 1307, 1311 (Fed. Cir. 2007).

1. "Virtual Function Processor" and "Function Processor Instructions"

Hypercom contends that CardSoft failed to present substantial evidence that Hypercom's terminals meet the "virtual function processor" and "function processor instructions" limitations. (JMOL Mot. at 5-6.) Hypercom also contends that the jury's finding that the limitations are present was against the great weight of the evidence and that a new trial is warranted. (NT Mot. at 4-5.) Both limitations are found in the two asserted claims. The Court found that "virtual

function processor" means "software which controls and/or selects general operations of a communication device." (Claim Construction Order at 20, Dkt. No. 251.) "Function processor instructions" means "a set of instructions that control operation of the communications device." (*Id.* at 25.)

CardSoft presented Joe Tipton Cole as its technical expert. Mr. Cole testified that he reviewed the source code for Hypercom's terminals, and found that the source code files containing the "virtual function processors" and "function processor instructions" were located in the "API interface directory." (6/5 a.m. Tr. 74:14-75:13.) Mr. Cole identified "USB_EXTR.C", "MDMEXTR.C", and "USBEXTR.C" as examples of the source code files that he found. (*Id.* at 74:14-20.) Although Mr. Cole's testimony focused more on the "virtual function processor" limitation, Mr. Cole was specifically asked if he found that both the "virtual function processor" and "function processor instructions" were met, which he confirmed:

> Q. And so it's your – it's your opinion that the virtual function processor and function processor instructions is found in the accused devices; is that correct?
>
> A. That's correct.

(*Id.* at 76:17-21.)

In its motion, Hypercom admits that Mr. Cole presented evidence that the "virtual function processor" is present in the Hypercom terminals. However, Hypercom argues that Mr. Cole failed to separately demonstrate that the "virtual function processor" and "function processor instructions" limitations are present, and that this failure violates the all elements rule. (JMOL Mot. at 5-6.) A straightforward reading of Mr. Cole's testimony shows that he expressed his opinion that both the "virtual function processor" and "function processor instructions" are satisfied. There is nothing in his testimony that suggests he relied on the same portions of the source code files to satisfy both limitations simultaneously. Instead, his testimony is reasonably

clear that the source code files contain both the "virtual function processor" code and the "function processor instructions" code. At no point did Hypercom cross-examine Mr. Cole regarding his opinion that the "function processor limitation" is satisfied. At best, Hypercom points to its own expert's testimony (Dr. Nettles) where he summarily states that Mr. Cole's conclusion is wrong, but provides no reasoning or analysis for that opinion. (*See* 6/6 p.m. Tr. 100-01.) Hypercom has not demonstrated any reason why it would be legally improper for the jury to credit Mr. Cole's testimony over Dr. Nettles'. Therefore, the Court finds that there is substantial evidence to support the jury's conclusion that the "virtual function processor" and "function processor instructions" limitations are present in Hypercom's terminals. Moreover, this finding is not against the weight of the evidence.

2. "Virtual Message Processor"

In its renewed motion for judgment as a matter of law, Hypercom argues that CardSoft failed to offer substantial evidence to support the jury's finding that Hypercom terminals have a "virtual message processor" because Mr. Cole failed to testify that the software he identified "assembles, disassembles, or compares messages." (JMOL Mot. at 3-4.) Hypercom raises the same issue in its motion for a new trial. (NT Mot. at 6-7.) The Court has found that a "virtual message processor" is "software implemented in the native code of the communications device that processes messages, including assembling, disassembling and/or comparing messages, for communication to and/or from a communications device." (Claim Construction Order at 19.)

Mr. Cole testified that the software constituting the "virtual message processor" is found in the source code files "USB.C" and "USB.H". (6/5 a.m. Tr. 79:11-80:1.) In the course of explaining how the "virtual function processor" calls the "virtual message processor," Mr. Cole explained his contention that that the identified files contain the code used to "construct, build up, and send out" messages. (*Id.* at 77:11-78:9 and 79:11-80:1.) Mr. Cole explained to the jury

how comments in the source code support his understanding that the identified source code meets the "virtual message processor" limitation as construed by the Court. (*Id.* at 82:19-84:2.)

There is no merit to Hypercom's argument that Mr. Cole simply relies on a physical USB port to satisfy this claim limitation. Although the files pointed to by Mr. Cole have the term "USB" in their names, Mr. Cole clearly relied on software code written by Hypercom to meet this limitation. The Court finds that there is substantial evidence to support the jury's finding that the Hypercom terminals meet the "virtual message processor" limitation and that the jury's finding is not against the weight of the evidence.

3. "Message Instruction Means"

Hypercom contends that CardSoft failed to present substantial evidence that Hypercom's terminals satisfy the "message instruction means" limitation on two separate grounds: (1) that CardSoft failed to show that a "message instruction means" in the Hypercom terminals provides directions for operation of the virtual message processor; and (2) that CardSoft failed to show that the "message instruction means" in the Hypercom terminals is identical or equivalent to the structure disclosed in the patent. (JMOL Mot. at 8-10.) Hypercom raises the same issues in its motion for a new trial. (NT Mot. at 10-11.)

The "message instruction means" is a means-plus-function limitation that is subject to the provisions of 35 U.S.C. § 112, paragraph 6. The claim term has the function of "providing directions for operation of the virtual message processor." (Claim Construction Order at 24.) The claim term's corresponding structure is disclosed in the patent at "13:29-14:2; 15:23-34; Figure 11 and Figure 8, and equivalents thereof." (*Id.*)

In its motion for judgment as a matter of law, Hypercom admits that Mr. Cole identified "USBINPUTREQ" and "USBOUTPUTREQ" as the data structures used to send messages.

(JMOL Mot. at 8 (citing 6/5 a.m. Tr. 77:10-22).) Mr. Cole goes on to testify that these structures perform the function specified in the Court's construction:

> So this section here defines something called the USBINPUTREQ, and this identifies the USBOUTPUTREQ. These are data structures that are used to send messages, and the – the parts of the system that I identify as the – as the virtual message processor employ these – these structures, [to] direct the operation of the virtual message processor.

(6/5 a.m. Tr. 77:16-22.) Although Hypercom's expert Dr. Nettles testified that he disagreed with Mr. Cole's understanding of these data structures and how they work (*see* 6/6 p.m. Tr. 99:8-25), the jury was certainly within its rights to credit Mr. Cole's testimony over Dr. Nettles' testimony. The Court finds that there is substantial evidence to support the jury's finding that the structures identified in the Hypercom terminals perform the function required by the "message instruction means." Moreover, the jury's finding in this regard is not against the weight of the evidence.

Turning to Hypercom's second basis for judgment as a matter of law (that the accused structure must be identical or equivalent to the disclosed structure), the Court finds that Mr. Cole testified that the structure he identified in the Hypercom terminals is identical or equivalent to the structures disclosed in the patent.

On direct examination, Mr. Cole identified the "USBINPUTREQ" and "USBOUTPUTREQ" data structures, and testified that they satisfied the "message instruction means" limitation. (*See* 6/5 a.m. Tr. 76:22-79:7.) On cross-examination, Hypercom's counsel examined Mr. Cole on this point:

> Q. All right. And for Hypercom, did you find the same literal structure?
>
> A. I'm sorry, the --
>
> Q. With regard to the Hypercom source code you looked at --
>
> A. Right. Right. I'm --

> Q. -- did you find the exact identical structure?
>
> A. No, no.
>
> Q. No. But your testimony is it's equivalent because it's communicating the same information in a different way?
>
> A. This is a way -- what I would say is that the patent used a very generic way of communicating what it was talking about, and that specifically what it was talking about for somebody writing a C program is what we saw in the examples that I showed today.
>
> Q. My question really was a little more basic. With regard to your testimony on Hypercom, you agree with me you are not telling this jury that Hypercom source code includes the identical structure, correct?
>
> A. I -- I don't recall seeing a -- in the examples that I showed, I don't recall, for instance, seeing anything concerning a line and column.
>
> Q. So again, just keep it real simple --
>
> A. Right.
>
> Q. -- you did not find in the Hypercom source code a structure that is identical to Figure 11?
>
> A. Correct.
>
> Q. All right. And again, your testimony is, well, it's not identical, but it's equivalent because it's communicating the same information in a different way?
>
> A. Yes.
>
> Q. And in your opinion, that's equivalent?
>
> A. I believe it is, yes.

(6/5 a.m. Tr. 152:8-153:17.) This questioning followed similar questioning by the same lawyer regarding the presence of the same limitation found in a co-defendant's product:

> Q. All right. Let's go to Figure 11. Did you find in the VeriFone source code a structure that was identical to Figure 11?

> A. I don't recall any pyramids or triangles drawn in the source code, so I'd have to say no.

(*Id.* at 151:18-22.)

It is clear from the context of the testimony that Mr. Cole was readily admitting that the exact illustration found in Figure 11 was not reproduced in either defendants' source code. This is not a surprising admission because the evidence showed that source code files are limited to containing text. This point was explained during CardSoft's redirect of Mr. Cole:

> Q. Okay. Is Figure 11 a -- a description of a data structure having fields?
>
> A. Yes.
>
> Q. Okay. Now, Figure 11 is also an image, isn't it?
>
> A. Yes.
>
> Q. Figure 11 is an image in the patent, but it discloses a data structure having fields; is that correct?
>
> A. That's correct.
>
> Q. Okay. And in the accused devices, you found a data structure having fields in every single accused device, didn't you?
>
> A. Yes, sir.
>
> Q. So that claim limitation is found literally. You have the structural limitation, which is identified in Figure 11, found in every accused device; isn't that correct?
>
> A. That's correct. And that was the way I phrased my opinion.
>
> Q. Now, you don't need to find the image in a patent figure and source code, do you?
>
> A. No.
>
> Q. That would be silly, wouldn't it?
>
> A. Well, I don't know, but it's not -- not the way that I have learned to do the work.

(6/5 p.m. Tr. 26:2-27:2.)

In considering the entirety of Mr. Cole's testimony, the Court finds that Mr. Cole's testimony provides substantial evidence to support the jury's finding that the "USBINPUTREQ" and "USBOUTPUTREQ" data structures are equivalent or identical to the structure disclosed by Figure 11 of the patent. The Court also concludes that this finding is not against the weight of the evidence. For the foregoing reasons, the Court denies Hypercom's motion for judgment as a matter of law and motion for a new trial as it relates to the "message instruction means."

4. "Virtual Machine Means" and "Emulatable"

Hypercom moves for judgment as a matter of law and a new trial on several grounds relating to the sufficiency of the evidence for the "virtual machine means" and "emulatable limitations." (JMOL Mot. at 11-15; NT Mot. at 10-14.) The asserted claims require a "virtual machine means," which is "a computer programmed to emulate a hypothetical computer for applications relating to transport of data." (Claim Construction Order at 14.) The asserted claims specify that "the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems." '945 Pat., cl. 1 and '683 Pat., cl. 1. The Court found that this limitation means that the "virtual machine means" is "capable of executing programs on different computers having incompatible hardware or operating systems." (Claim Construction Order at 17.)

First, Hypercom argues that CardSoft failed to present any evidence that Hypercom's terminals have a "virtual machine means," or that the "virtual machine means" satisfies the "emulatable" limitation. (JMOL Mot. at 11.) Hypercom argues that Mr. Cole skipped over the "virtual machine means" limitation and improperly merged the limitation with the "emulatable" limitation. (*Id.*)

Mr. Cole testified that the "emulatable" limitation is met by the Hypercom terminals because Hypercom's source code implementing the "virtual machine means" is designed to be executable on the Intel/Marvell PXA255 processor and the Zilog ZA9 processor, which are different incompatible processors. (6/5 a.m. Tr. 84:3-85:13.) Although Mr. Cole offered in short form his opinion that the Hypercom terminal software has a "virtual machine means," Hypercom did not cross-examine[2] him on the basis for his opinion, and therefore the jury was free to accept or reject this opinion evidence. *See Symbol Techs., Inc. v. Opticon, Inc.*, 935 F.3d 1569, 1576 (Fed. Cir. 1991). The Court finds that there is no merit to Hypercom's argument that Mr. Cole "skipped" or improperly "merged" the "virtual machine means" limitation.

Next, Hypercom contends that CardSoft failed to present sufficient evidence that the "virtual machine means' is "capable of <u>executing</u> programs on different computers having incompatible hardware or operating systems." (JMOL Mot. at 11-13.) Hypercom argues that "Cardsoft's counsel did not even ask Cole whether the alleged virtual machine means on Hypercom terminals actually executes programs" and then states that "[o]n cross-examination, Cole did assert that the virtual machine means executes programs but made no effort explain which portion of the virtual machine means does so or explain how." (JMOL Mot. at 13.)

On direct examination, Mr. Cole stated that it was his opinion that the "emulatable" limitation was met. (*See* 6/5 a.m. Tr. 84:3-85:13.) To the extent that opinion was not clear, Hypercom's cross-examination cleared up that point:

> Q. So it's your testimony that on the Hypercom terminals, the virtual machine means that you've testified to is executing application programs?

---

[2] Hypercom's briefing suggests that Mr. Cole offered no testimony concerning the "virtual machine means." The Court observes that some of Hypercom's cross-examination questioning strongly suggests otherwise. (*See, e.g.*, 6/5 a.m. Tr. 126:15-18 ("the virtual means that you've testified to . . . .").)

A. Yes.

(6/5 a.m. Tr. 126:15-18.) Hypercom does not point the Court to where it asked Mr. Cole to explain the basis for this opinion, or otherwise pointed out to the jury that nothing in particular was identified to be the virtual machine means or how it functions. The Court finds that there is no merit to Hypercom's "executing programs" argument.

Hypercom argues that there is insufficient evidence that the "emulatable" limitation is satisfied because Mr. Cole did not analyze whether the "virtual machine means" in the Hypercom terminals is capable of executing programs having an incompatible operating system. (JMOL Mot. at 13-14.) The Court finds that this argument has no merit in light of the Court's construction. The "emulatable" limitation only requires showing that the "virtual machine means" is capable of executing programs on different computers having 1) incompatible hardware or 2) incompatible operating systems. CardSoft clearly presented evidence that the Hypercom "virtual machine means" is capable of executing programs on different computers having incompatible hardware (*e.g.*, different microprocessors).

Lastly, Hypercom argues that CardSoft failed to show that Hypercom's terminals have a "virtual machine means" that is "capable of executing programs on different computers having incompatible hardware or operating systems." (JMOL Mot. at 14-15.) Hypercom's motion has required the Court to review Mr. Cole's testimony concerning the "virtual machine means" and "emulatable" limitations multiple times. As recounted above, Mr. Cole testified that it was his opinion that both limitations were met, and he provided his explanation for those opinions. Hypercom alleges that this testimony was deficient, but does not point to anywhere in the record where it pointed out to the jury the failings that it now complains of in its motion, either on cross-examination or through its rebuttal evidence. After considering all of Hypercom's arguments, the Court finds that there is substantial evidence to support the jury's finding of

infringement for the "virtual machine means" and the "emulatable" limitations. The Court also finds that the jury's conclusion that the "virtual machine means" and "emulatable" limitations are present in Hypercom's terminals is not against the weight of the evidence.

B. Exclusion of Certain Opinions of Dr. Nettles

Hypercom moves for a new trial on the basis that the Court improperly struck paragraphs 51, 52, and 64 of Dr. Scott Nettles' expert report (Hypercom's technical expert). (NT Mot. at 3-4.) The Court's order striking these portions of Dr. Nettles' report set forth in full the opinions that were stricken, as well as the Court's reasoning for excluding them. (*See* Mem. Order, Dkt. No. 372.) After considering the arguments of the parties, the Court is convinced that the exclusion of those opinions was proper, and that a new trial is not warranted.

CONCLUSION

For the foregoing reasons, Hypercom's Renewed Motion for Judgment as a Matter of Law (Dkt. No. 426) is **DENIED,** and Hypercom's Motion for New Trial (No. 427) is **DENIED**.

SIGNED this 27th day of October, 2013.

ROY S. PAYNE
UNITED STATES MAGISTRATE JUDGE

Judgment, Dated October 30, 2013

**IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

CARDSOFT (ASSIGNMENT FOR THE BENEFIT OF CREDITORS), LLC,	§	
Plaintiff,	§	
v.	§	Case No. 2:08-CV-98-RSP
VERIFONE SYSTEMS, INC.; VERIFONE INC.; and HYPERCOM CORP.,	§	
Defendants.	§	

JUDGMENT

This civil action came before the Court for a trial by jury. The issues have been tried and the jury rendered a verdict on June 8, 2012. (Dkt. No. 389.) The Court **ORDERS AND ENTERS JUDGMENT** between Plaintiff CardSoft (Assignment for the Benefit of Creditors), LLC (hereinafter "CardSoft") and Defendants VeriFone Systems, Inc. (hereinafter "VeriFone Systems"), VeriFone Inc. (hereinafter "VeriFone"), and Hypercom Corp. (hereinafter "Hypercom") AS FOLLOWS:

1. VeriFone Systems, VeriFone, and Hypercom are found to infringe claim 11 of United States Patent No. 6,934,945 ("the '945 Patent").

2. The infringement of claim 11 of the '945 Patent was not willful.

3. Claim 11 of the '945 Patent is found not invalid as anticipated.

4. VeriFone Systems, VeriFone, and Hypercom are found to infringe claim 1 of United States Patent No. 7,302,683 ("the '683 Patent").

5. The infringement of claim 1 of the '683 Patent was not willful.

6. Claim 1 of the '683 Patent is found not invalid as anticipated.

7. CardSoft is awarded damages for VeriFone Systems's and VeriFone's infringement in the amount of $13,148,958.00, at a royalty rate of $3.00 per infringing unit.

8. CardSoft is awarded damages for Hypercom's infringement in the amount of $2,245,863.00, at a royalty rate of $3.00 per infringing unit.

9. CardSoft is awarded pre-judgment interest, post-judgment interest, and costs.

10. All relief not specifically granted herein is **DENIED**, subject to **SEVERANCE** of CardSoft's request for an ongoing royalty as ordered in the Court's Memorandum Order (Dkt. No. 482).

IT IS SO ORDERED.

SIGNED this 30th day of October, 2013.

ROY S. PAYNE
UNITED STATES MAGISTRATE JUDGE

Case: 14-1135 Document: 32 Page: 163 Filed: 02/18/2014

Patent No. 6,934,945 , Dated August 23, 2005

US006934945B1

(12) **United States Patent**
Ogilvy

(10) **Patent No.: US 6,934,945 B1**
(45) **Date of Patent: Aug. 23, 2005**

(54) **METHOD AND APPARATUS FOR CONTROLLING COMMUNICATIONS**

(75) Inventor: **Ian Charles Ogilvy**, Manly (AU)

(73) Assignee: **Cardsoft, Inc.**, San Mateo, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/381,143**

(22) PCT Filed: **Mar. 16, 1998**

(86) PCT No.: **PCT/AU98/00173**

§ 371 (c)(1),
(2), (4) Date: **Oct. 22, 1999**

(87) PCT Pub. No.: **WO98/41918**

PCT Pub. Date: **Sep. 24, 1998**

(30) **Foreign Application Priority Data**

Mar. 19, 1997 (AU) PO9896

(51) **Int. Cl.**[7] **G06F 17/00**

(52) **U.S. Cl.** 718/1; 719/313; 719/310; 719/315; 719/316

(58) **Field of Search** 718/1; 719/310, 719/313, 315, 316; 703/23, 26, 27; 709/2, 201, 208, 219, 200; 713/201; 395/500

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,336,870 A	* 8/1994	Hughes et al.	....	235/379
5,479,643 A	12/1995	Bhaskar		
5,553,291 A	* 9/1996	Tanaka et al.	....	718/1
5,590,281 A	12/1996	Stevens		
5,708,838 A	* 1/1998	Robinson	....	709/202
5,745,886 A	* 4/1998	Rosen	....	705/39
5,926,631 A	* 7/1999	McGarvey	....	703/23
5,931,917 A	* 8/1999	Nguyen et al.	....	709/203
5,935,249 A	* 8/1999	Stern et al.	....	713/201
6,003,065 A	* 12/1999	Yan et al.	....	709/201
6,199,160 B1	* 3/2001	Echensperger et al.	.....	713/100
6,308,317 B1	* 10/2001	Wilkinson et al.	....	717/139

FOREIGN PATENT DOCUMENTS

AU	2200088	9/1988
EP	0456249	11/1991
EP	0634718	1/1995
WO	9212480	7/1992
WO	9305599	3/1993
WO	9410628	5/1994

OTHER PUBLICATIONS

Kiran Kumar Toutireddy, "A Testbed for Fault Tolerant Real–Time Systems", Sep. 1996, pp. 1–120.*
Ray Hookway, "Digital FX!32", Feb. 1997, IEEE, pp. 37–42.*
Hamilton, "Java and the Shift to Net–Centric Computing", P–31–39; Aug. 1996, Computer.

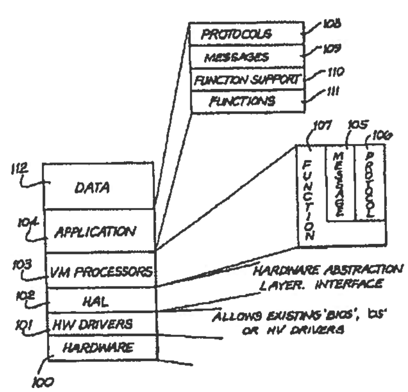
* cited by examiner

Primary Examiner—David Wiley
Assistant Examiner—Phuoc Nguyen
(74) *Attorney, Agent, or Firm*—Fleshner & Kim, LLP

(57) **ABSTRACT**

The present invention relates to preparing and processing information to be communicated via a network or to or from other data carriers. For implementation of a novel "virtual machine" of the present invention, a minimal amount of hardware is required. Prior art virtual machines tend to slow down operation of the device as they interface between an application program and device drivers. The novel virtual machine incorporates a virtual message processing means that is arranged to construct, deconstruct and compare messages and applied in the native code of the processor. The message instruction means directs and controls the message processor. Similarly, a protocol processor means governs and organs communications, under the direction of a protocol instruction means in the application. These elements of the novel virtual machine increase the speed and efficiency and allow implementation of a practical device for use in communications, able to be implemented on different hardware having different BIOS/OS.

17 Claims, 12 Drawing Sheets



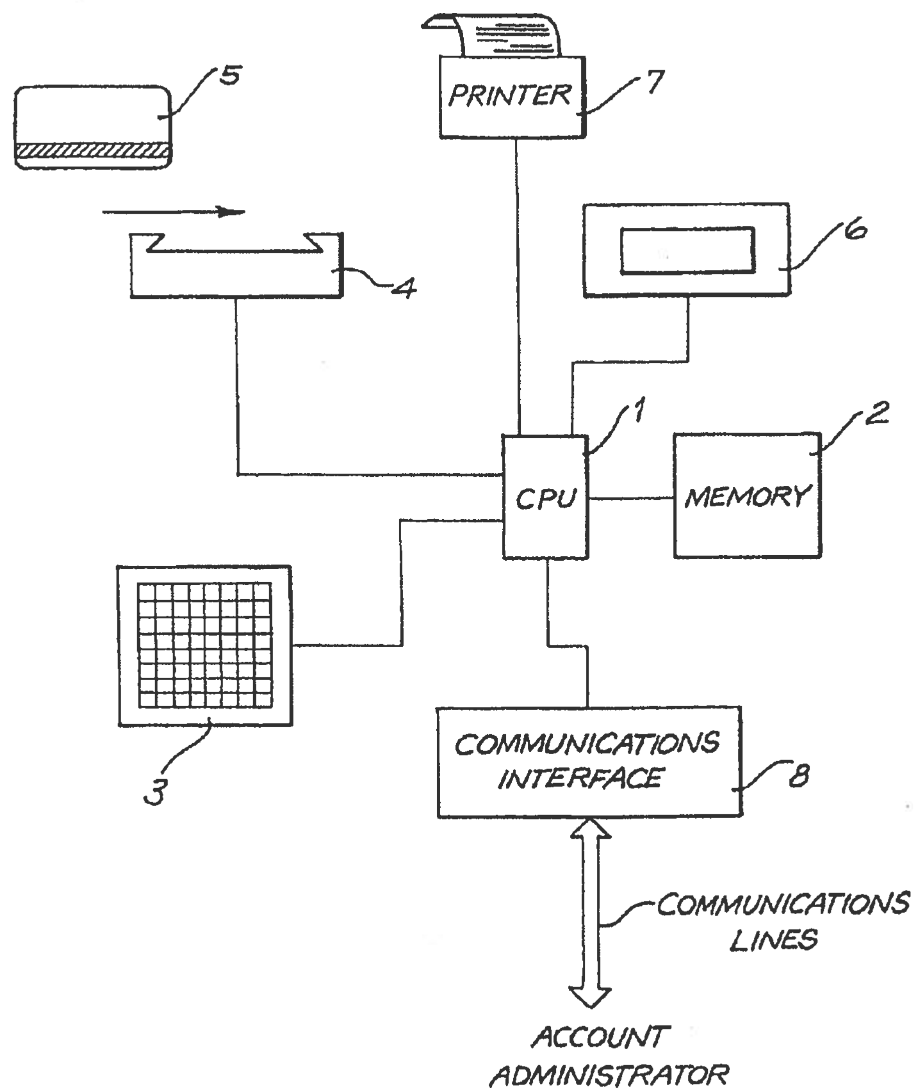

5
PRINTER
7
6
4
1
2
CPU
MEMORY
3
COMMUNICATIONS
INTERFACE
8
COMMUNICATIONS
LINES
ACCOUNT
ADMINISTRATOR

FIG. 1

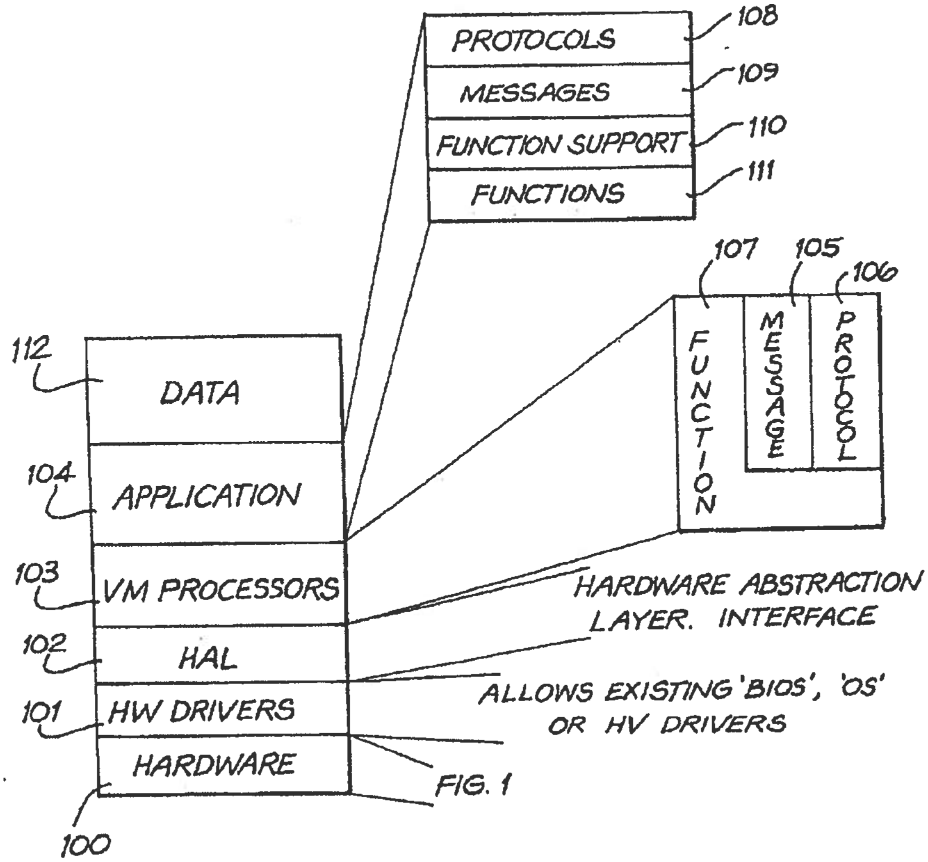

PROTOCOLS
108
MESSAGES
109
FUNCTION SUPPORT
110
FUNCTIONS
111
107
105
106
FUNCTION
MESSAGE
PROTOCOL
112
DATA
104
APPLICATION
103
VM PROCESSORS
102
HAL
101
HW DRIVERS
HARDWARE
100
HARDWARE ABSTRACTION LAYER. INTERFACE
ALLOWS EXISTING 'BIOS', 'OS' OR HV DRIVERS
FIG. 1

FIG. 2

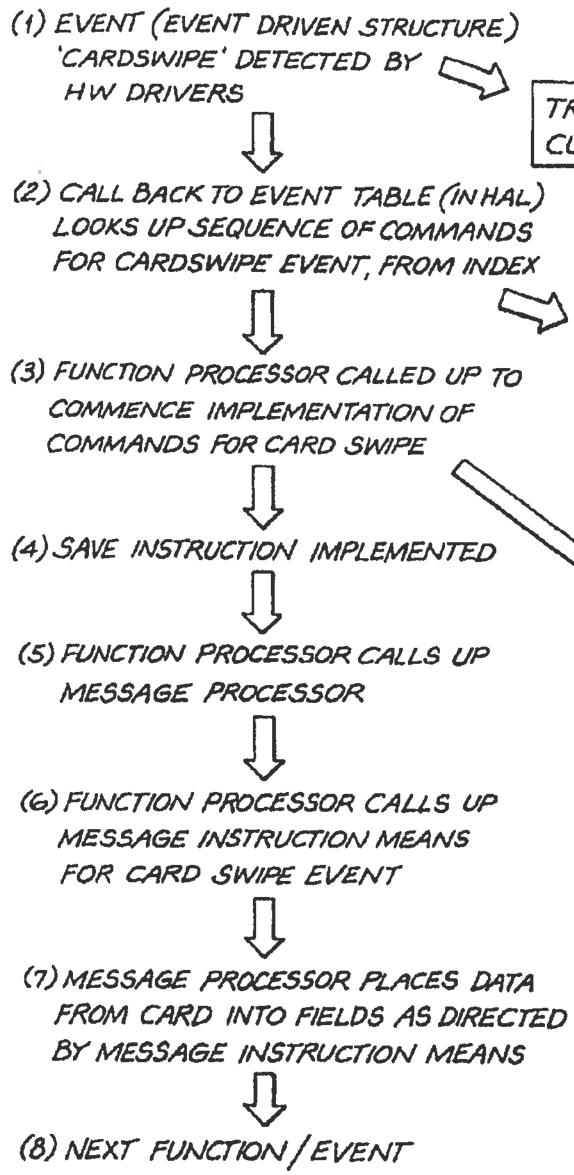


FIG. 3a

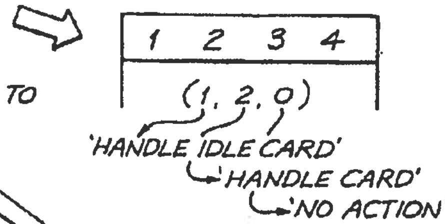


FIG. 3b

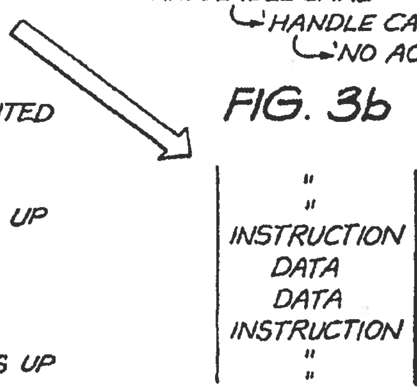


FIG. 3c

FIG. 3

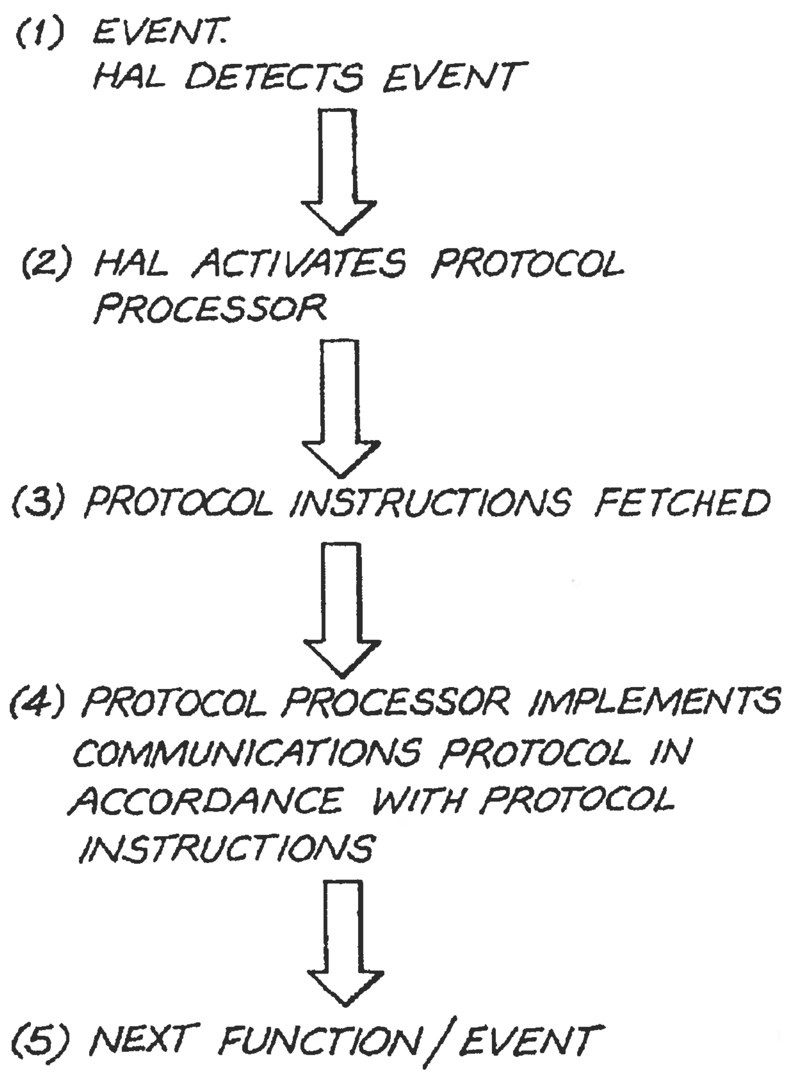

(1) EVENT.
HAL DETECTS EVENT
(2) HAL ACTIVATES PROTOCOL PROCESSOR
(3) PROTOCOL INSTRUCTIONS FETCHED
(4) PROTOCOL PROCESSOR IMPLEMENTS COMMUNICATIONS PROTOCOL IN ACCORDANCE WITH PROTOCOL INSTRUCTIONS
(5) NEXT FUNCTION/EVENT

FIG. 4

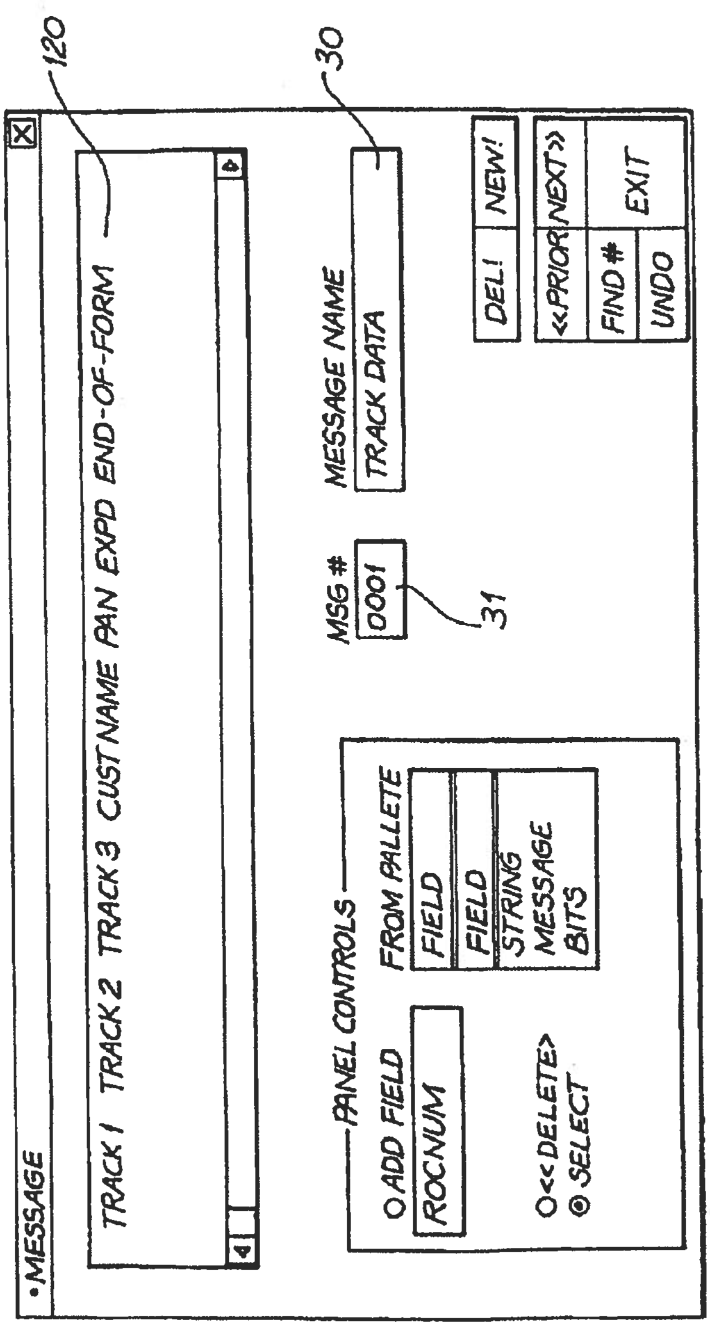

MESSAGE
120
TRACK 1 TRACK 2 TRACK 3 CUST NAME PAN EXPD END-OF-FORM
30
MESSAGE NAME
TRACK DATA
MSG #
0001
31
DEL!
NEW!
<<PRIOR
NEXT>>
FIND #
EXIT
UNDO
PANEL CONTROLS
ADD FIELD
ROCNUM
FROM PALLETE
FIELD
FIELD
STRING
MESSAGE
BITS
<<DELETE>
SELECT

FIG. 5

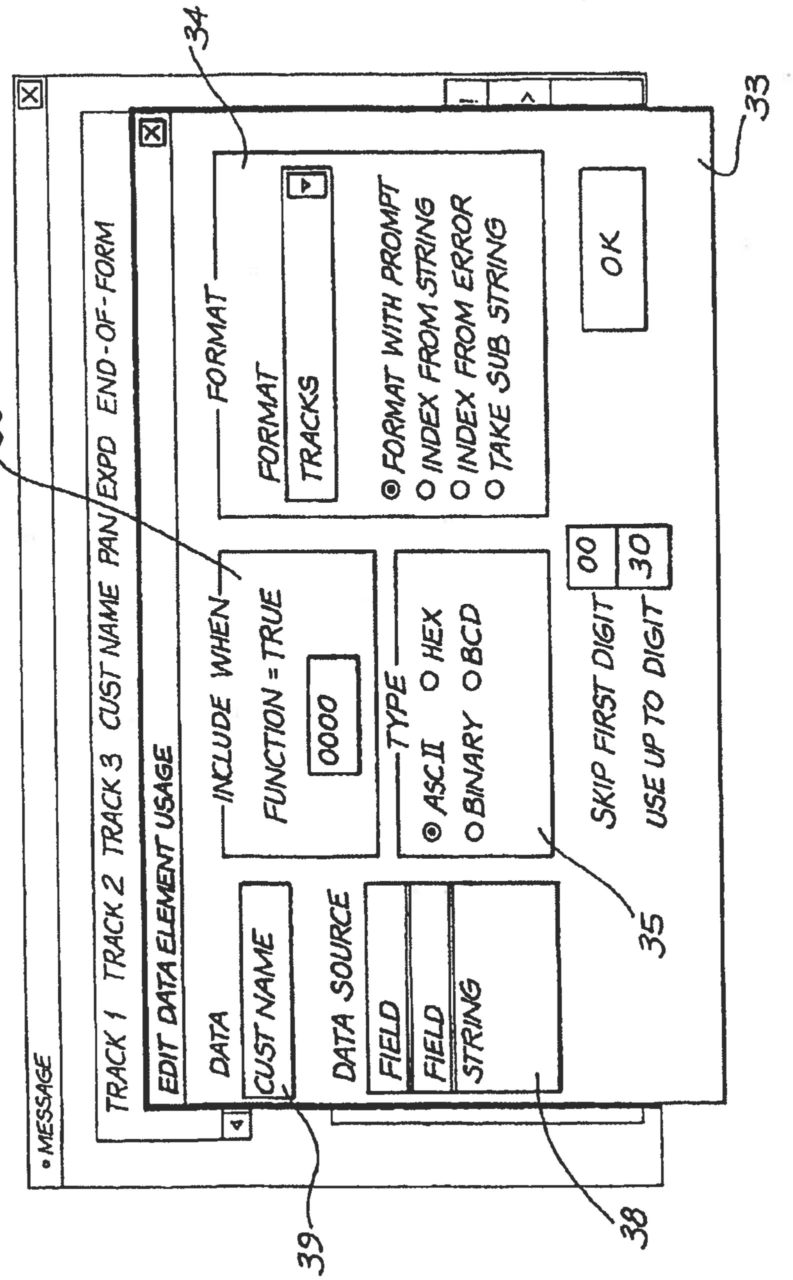

MESSAGE
TRACK 1 TRACK 2 TRACK 3 CUST NAME PAN EXPD END-OF-FORM
EDIT DATA ELEMENT USAGE
DATA
CUST NAME
DATA SOURCE
FIELD
FIELD
STRING
INCLUDE WHEN
FUNCTION = TRUE
0000
TYPE
ASCII
HEX
BINARY
BCD
FORMAT
FORMAT
TRACKS
FORMAT WITH PROMPT
INDEX FROM STRING
INDEX FROM ERROR
TAKE SUB STRING
OK
SKIP FIRST DIGIT
00
USE UP TO DIGIT
30
33
34
35
36
38
39

FIG. 6

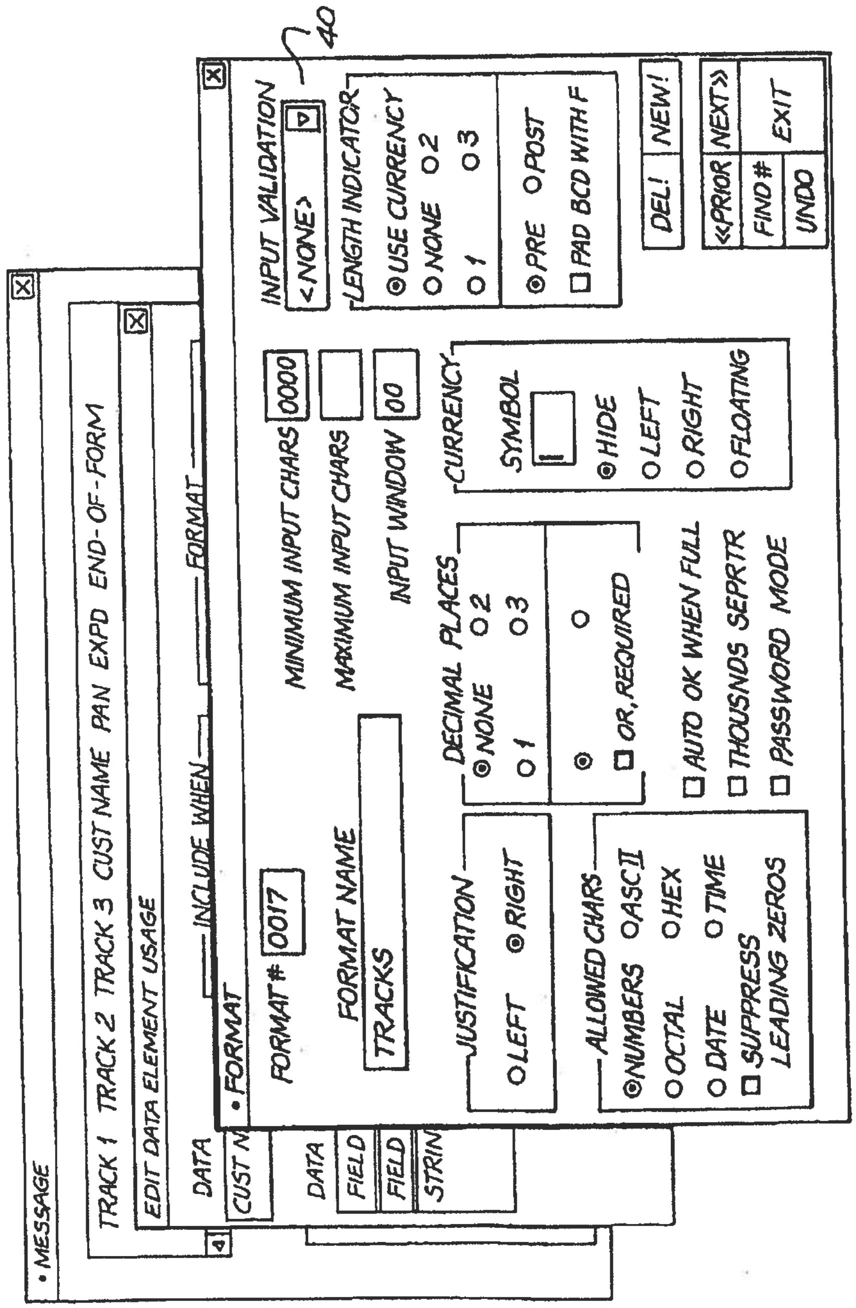

40
• MESSAGE
TRACK 1 TRACK 2 TRACK 3 CUST NAME PAN EXPD END-OF-FORM
EDIT DATA ELEMENT USAGE
INCLUDE WHEN
FORMAT
DATA
CUST N
DATA
FIELD
FIELD
STRIN
• FORMAT
FORMAT # 0017
FORMAT NAME
TRACKS
MINIMUM INPUT CHARS 0000
MAXIMUM INPUT CHARS
INPUT WINDOW 00
INPUT VALIDATION
<NONE>
LENGTH INDICATOR
USE CURRENCY
NONE 2
1 3
PRE POST
PAD BCD WITH F
DEL! NEW!
<<PRIOR NEXT>>
FIND # EXIT
UNDO
JUSTIFICATION
LEFT RIGHT
DECIMAL PLACES
NONE 2
1 3
OR, REQUIRED
CURRENCY
SYMBOL
HIDE
LEFT
RIGHT
FLOATING
ALLOWED CHARS
NUMBERS ASCII
OCTAL HEX
DATE TIME
SUPPRESS LEADING ZEROS
AUTO OK WHEN FULL
THOUSNDS SEPRTR
PASSWORD MODE

FIG. 7

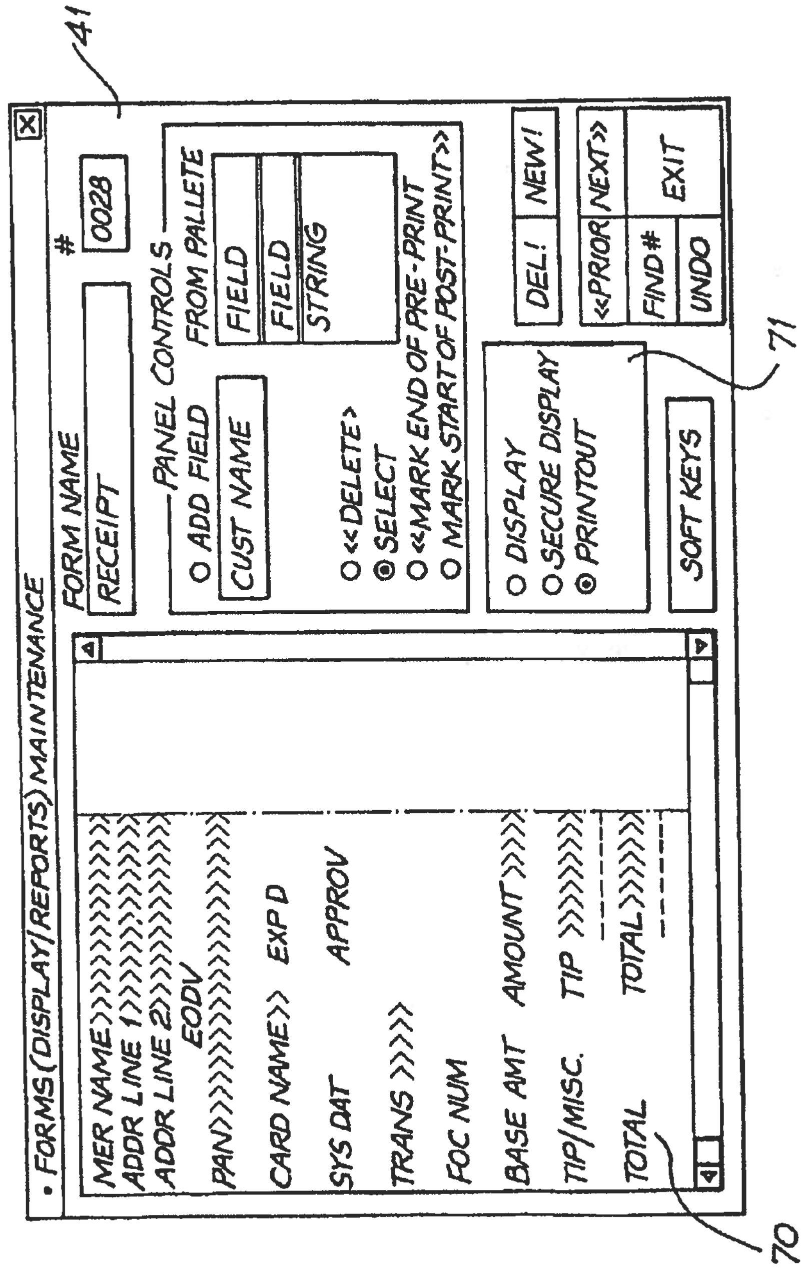

41
FORMS (DISPLAY/REPORTS) MAINTENANCE
FORM NAME
RECEIPT
#
0028
PANEL CONTROLS
ADD FIELD
CUST NAME
FROM PALLETE
FIELD
FIELD
STRING
<<DELETE>
SELECT
<<MARK END OF PRE-PRINT
MARK START OF POST-PRINT>>
DEL!
NEW!
<<PRIOR
NEXT>>
FIND#
UNDO
EXIT
DISPLAY
SECURE DISPLAY
PRINTOUT
SOFT KEYS
71
MER NAME
ADDR LINE 1
ADDR LINE 2
EODV
PAN
CARD NAME
EXP D
SYS DAT
APPROV
TRANS
FOC NUM
BASE AMT
AMOUNT
TIP/MISC.
TIP
TOTAL
TOTAL
70

FIG. 8

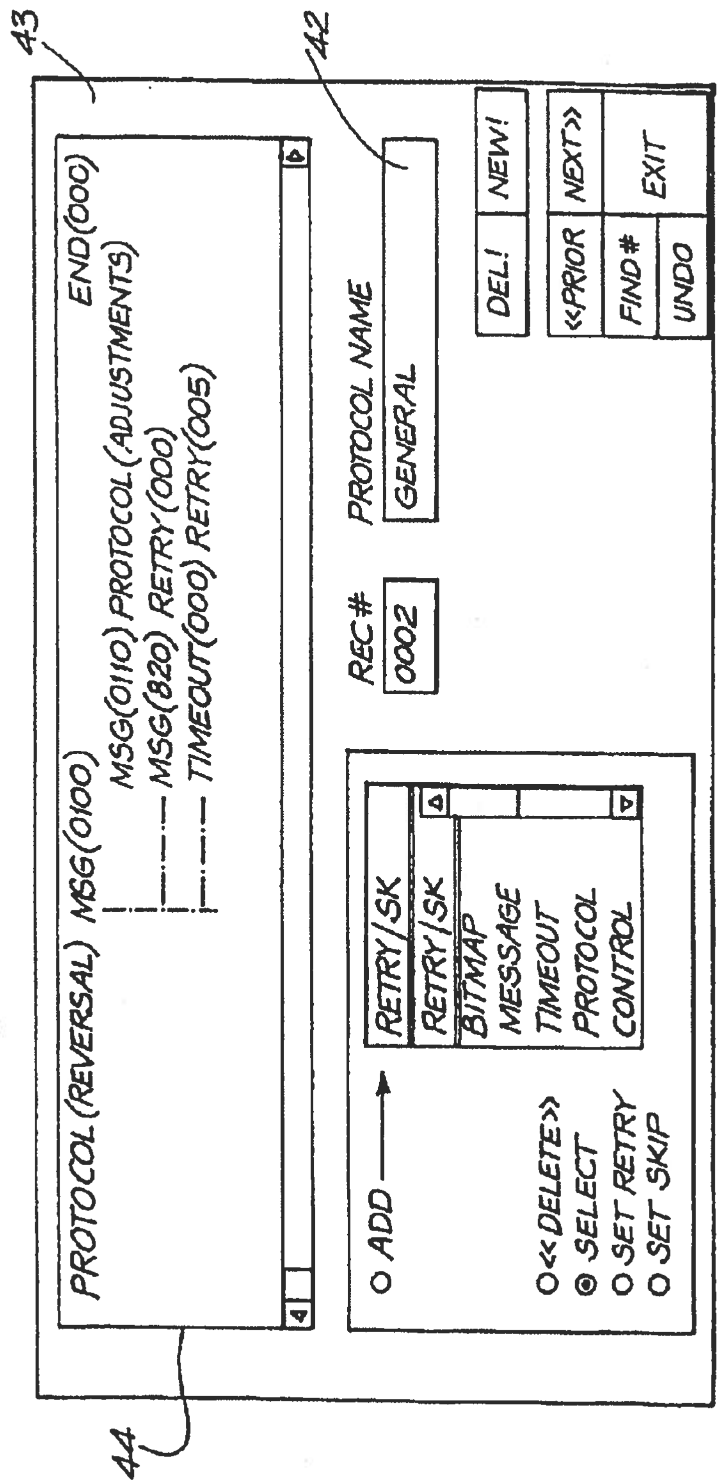

43
42
PROTOCOL (REVERSAL) MSG(0100)
MSG(0110) PROTOCOL (ADJUSTMENTS)
MSG(820) RETRY(000)
TIMEOUT(000) RETRY(005)
END(000)
REC#
0002
PROTOCOL NAME
GENERAL
DEL!
NEW!
<<PRIOR
NEXT>>
FIND#
EXIT
UNDO
ADD
RETRY/SK
RETRY/SK
BITMAP
MESSAGE
TIMEOUT
PROTOCOL
CONTROL
<<DELETE>>
SELECT
SET RETRY
SET SKIP
44

FIG. 9

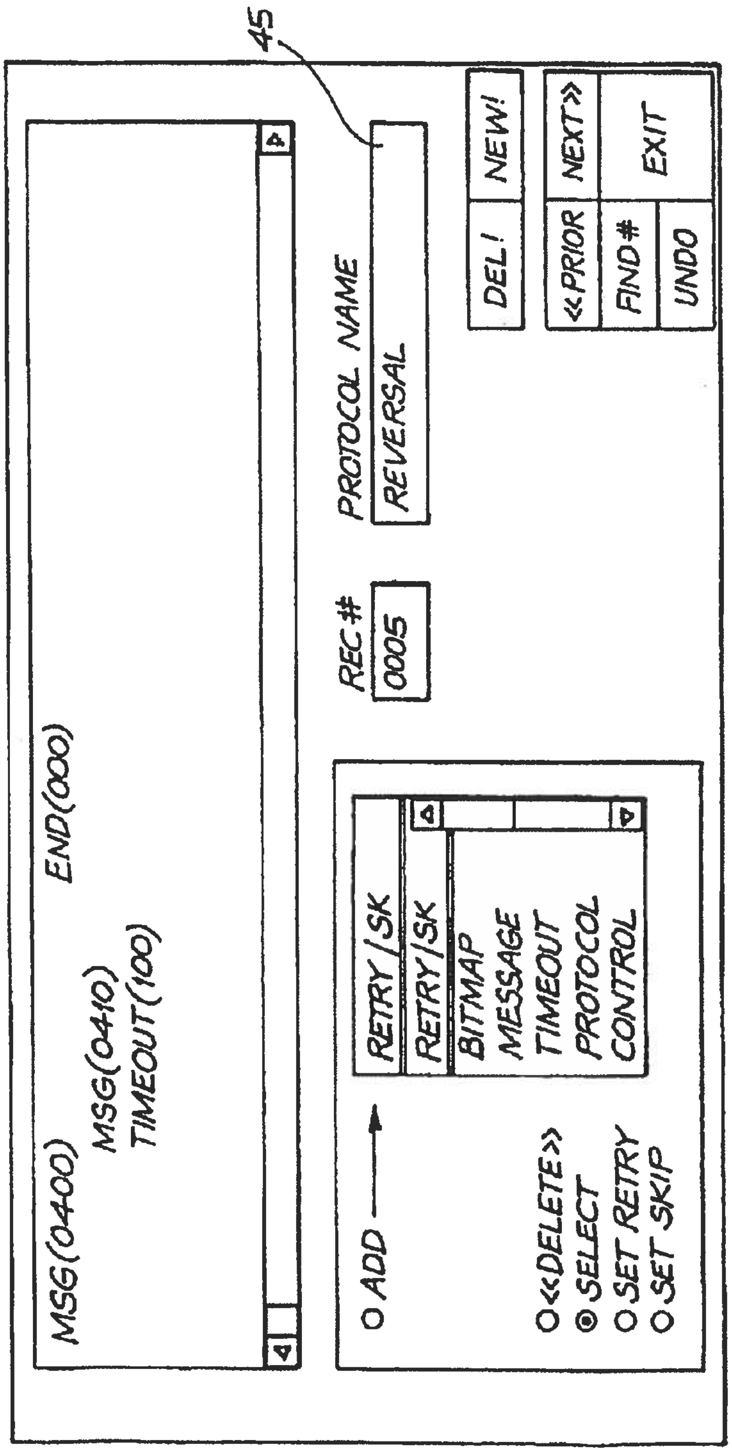

45
MSG(0400)
MSG(0410)
TIMEOUT(100)
END(000)
PROTOCOL NAME
REVERSAL
REC #
0005
DEL!
NEW!
<<PRIOR
NEXT>>
FIND #
UNDO
EXIT
ADD
<<DELETE>>
SELECT
SET RETRY
SET SKIP
RETRY/SK
RETRY/SK
BITMAP
MESSAGE
TIMEOUT
PROTOCOL
CONTROL

FIG. 10

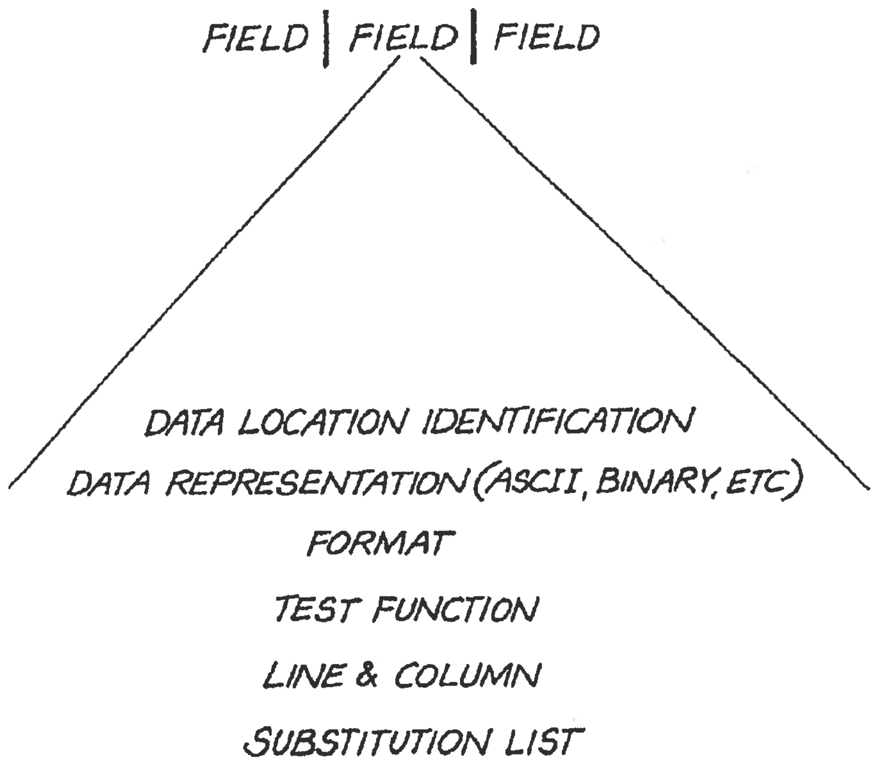

FIELD | FIELD | FIELD
DATA LOCATION IDENTIFICATION
DATA REPRESENTATION (ASCII, BINARY, ETC)
FORMAT
TEST FUNCTION
LINE & COLUMN
SUBSTITUTION LIST

FIG. 11

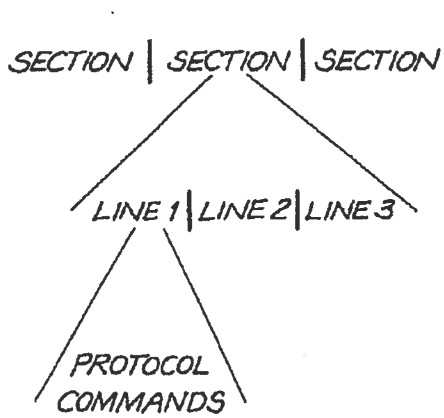


FIG. 12a

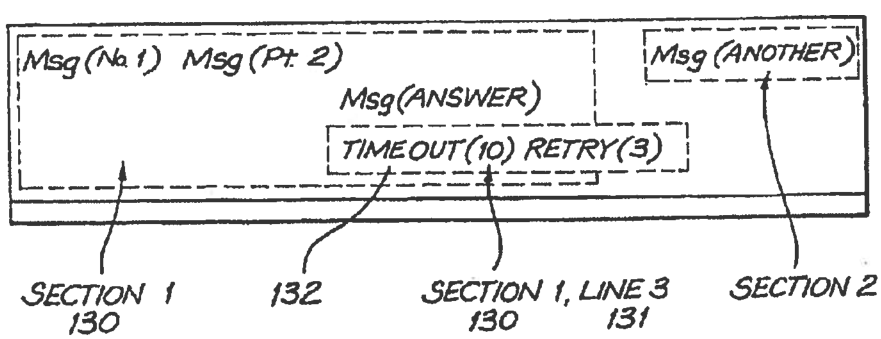


FIG. 12b

METHOD AND APPARATUS FOR CONTROLLING COMMUNICATIONS

From a first, general aspect, the present invention relates to a method and apparatus for preparing and processing information to be sent or received via a network. A network in this instance may be implemented as data carried either over communications lines and/or stored on smart cards (or other data carriers) and physically transported.

From a second, more specific aspect, the present invention relates to a method and apparatus for controlling remote payment transactions, particularly, but not exclusively, for controlling remote payment transactions where a persons account is credited and/or debited from a remote location in exchange for goods/services cash or credit, or where account information is accessed remotely to enable approval of a transaction.

Devices for carrying out remote payment transactions are well-known. These "payment terminals" include EFTPOS, credit card payment terminals, etc.

The most common function of payment terminals is to remotely access a persons account information and either carry out a transaction, such as crediting or debiting the account, or, particularly in the case of credit card payment terminals, to check the users account to ensure that there are sufficient funds to cover a transaction. Note that although credit card terminals do not necessarily remotely credit or debit the users account (the credit/debit transaction usually being carried out by a separate paper bill trail) and merely provide the information that the users account is sufficient to cover the transaction, such payment terminals still fall within the ambit of the present invention and the term "transaction" as used herein includes the operation of remotely checking the users account to "ok" a transaction.

A payment terminal may, for example, provide for the following basic operations:

(1) Input of information which is required to enable access to a customers account. The information is most often read from a magnetic stripe on a credit card or bank card or the like, or a smart card. In addition to reading details from a card a personal identification number (PIN) or the like code may also be required.

(2) Obtain access to the customers account. This is usually done by remote communication with a processing device holding the person's account data, usually on bank premises and remote from the payment terminal. Usually, information on the customers account input to the payment terminal will need to be transmitted for verification and to enable access to the account. Also a money amount will usually need to be input to the payment terminal and transmitted over the communications line. At least some and perhaps all of the transmitted data may be encrypted for security purposes and the payment terminal is therefore, in such a case, required to have means (3) providing encryption.

(4) The payment terminal may need to be able to receive communications over the remote line from the processor accessing the customers account, ie. to provide an "answer" to the payment device regarding the user transaction. The answer may include information that an account debit/credit has taken place (eg. EFTPOS) or merely an approval that the customer has enough money in his account to enable a transaction (some credit card payment terminals). Again, this transmitted information may be encrypted and, if so, will require translation (5) in the payment terminal.

(6) To provide an indication that the transaction request is approved or that a transaction has occurred, by display or printer, for example. Displays may also be required to prompt an operator or customer to input information, e.g., input your PIN "Input Amount".

There are many different brands of payment terminal, utilising many different software and hardware arrangements. This gives rise to a number of problems.

Any account acquirer (eg. bank) will generally have their own operating requirements as to how remote payment trans-actions will be handled. The account acquirer may purchase a series of payment terminals which have been configured by a manufacturer to the acquirer's requirements. These payment terminals will then be licensed or rented or more often supplied at no charge to merchants (e.g., retail stores, garages, restaurants). Multiple account acquirers may require access to their customers accounts via a single payment terminal. That is, one particular merchant may operate payment terminals which provide access to customers accounts at other account acquirers (e.g., other banks). Because of different requirements of different acquirers for handling of remote payment transactions, the payment terminal must be arranged to operate to satisfy the different requirements.

The terminal owner (often a principle acquirer) will have the terminal appropriately arranged and programmed by the terminal manufacturer to satisfy the requirements of all account acquirers utilising the terminal. Payment terminals may need to contain several programs and select the appropriate program depending on the card to be processed or on an operator selection.

It is often the case that the terminal owner may need to have the operation of the payment device amended to, for example, enable it to operate for an additional account acquirer, or to satisfy changed requirements for a particular account acquirer. Because of the different hardware/software architectures available, any operational alterations generally the require the input of the terminal supplier or manufacturer. The supplier/manufacturer will be required to reprogram the terminal or amend the hardware in order to carry out the alterations and they will usually be the only person who has the appropriate knowledge. The terminal owner is thus tied to the particular supplier/manufacturer of the particular brand of payment terminal.

It is often the case that, the terminal owner may over time obtain different brands from different manufacturers and for operational alterations may need to return the particular brand to each separate manufacturer. Over time, manufacturers may go out of business, in which case the payment terminals made by that particular manufacturer may be unsupported and any alteration may be difficult to achieve, or at least will require the input of a skilled person having detailed knowledge of the programming and/or hardware of the redundant manufacturer's devices.

Being tied to a particular manufacturer for a particular brand therefore causes cost, time and trouble when any operational alterations are required. There is therefore a reluctance to carry out operational alterations, which sometimes means that requirements of various account acquirers are not fully satisfied. When an operational alteration does have to be carried out, it is costly. If a manufacturer goes out of business, the terminal owner may be left with nobody to alter the operation of his payment terminals, or indeed maintain the payment terminals. The present system is costly and inflexible.

A payment terminal device usually includes a microprocessor and a number of peripheral units (e.g., card reader, display, printer, communications interface, etc) controlled by the processor. A payment terminal device usually com-

prises hardware, an operating system or a BIOS and is ready to accept an application for that arrangement. Or the device may be supplied with an interpreter to accept the applications.

To alter the operation of payment terminals, a new application must be created. This can be time consuming, costly and as the programming will be different for different types of devices, which may have different hardware arrangements as well, and must be carried out separately for each different type of device (i.e., different reprogramming operations must be carried out for different devices even where the same operational alterations may be required).

The programming alterations are not "portable" between different types of devices.

The most time critical aspects of operation of a remote payment terminal involve the building up and breaking down of "messages" and the formulation and operation of communications. By "messages" is meant, for example, information data which is required to be input to the device or communicated or displayed in order to enable carrying out of a remote payment transaction, and includes information to be communicated to the bank, e.g., customers card number, customers PIN, amount of transaction, etc; displayed information such as "Please Input Amount"; information to be read from a customers magnetic stripe card or smart card and manipulated by the device e.g., card number, expiry date, etc. The operation of payment terminals is greatly concerned with the collection, rearrangement and communication of this message data to enable a remote payment transaction.

In conventional devices, each time a message is constructed or deconstructed, the operation of the machine will be handled by the application program. To change operation of the machine, the application must be changed. This is laborious, and gives rise to problems, as discussed above.

The technique of creating a virtual processor (or in this case microprocessor) is well known and referred to as an interpreter. This allows programs to operate independent of processor. With the newer technique of also creating virtual peripherals then the whole is referred to as a "virtual machine".

A virtual machine is computer programmed to emulate a hypothetical computer. Different incompatible computers may be programmed to emulate the same hypothetical computer. Any computer programmed to emulate the hypothetical computer will thus be capable of executing programs for the virtual computer. This creates a complete portable environment for program operations.

A problem with virtual machines is emulation is slower than normal program execution. For some applications this performance penalty is a significant problem.

The above problems and disadvantages which have been discussed specifically in relation to devices configured to process payment transactions also would apply to devices configured to prepare and process any information to be sent or received via a network, not restricted to payment transaction information.

From a first aspect the present invention provides a communications device which is arranged to process messages for communications, comprising a virtual machine means which includes a virtual function processor and function processor instructions for controlling operation of the device, and a virtual message processor which is arranged to be called by the function processor and which is arranged to carry out the task of assembling, disassembling and comparing messages, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task.

"Communications" includes transport of data via a data carrier such as a smart card.

By messages we mean a sequence of data comprising usually a plurality of information fields to be communicated.

The message processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The message processor instructions are preferably virtual instructions to be expressed only in the language defined by the message processor means- and thus never requiring translation to any real hardware processor.

The message processor means in at least a preferred embodiment provides two specific advantages over conventional arrangements

1) Faster Operation. The processor (executing as native code) operates at full microprocessor speed overcoming the problem of slow emulation speed for message related functions.

2) Faster, simpler programming. The instructions for the message processor preferably consist of actual message "descriptions". The programmer need only describe the message content, all data conversion, manipulation and processing is automatically performed based on the message description. This is a more intuitive and compartmentalised approach which preferably leads to faster programming with less errors.

The protocol processor means is preferably a program module the specific function of which is to control and select the sequence of message processor operations in relation to messages received and transmitted.

The protocol processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The protocol processor instructions are virtual instructions expressed only in the language defined by the protocol processor means and thus never requiring translation to any real hardware processor. The protocol processor means provides two specific advantages over conventional arrangements:

1) Faster Operation. The processor (executing as native code) preferably operates at full microprocessor speed overcoming the problem of slow emulation speed for protocol related functions.

2) Faster, simpler programming. The instructions for the protocol engine preferably consist of an actual diagram of the message flow. To change message flow or sequence, the programmer can modify an intuitive diagram, all multiprocessing and other complications are handled automatically. This more intuitive and compartmentalised approach leads to faster programming with less errors.

In a preferred embodiment, therefore, a device in accordance with the present invention includes a virtual machine including virtual processors which are specifically arranged to control message construction, deconstruction, comparison and to control the communication of information, both for reception from a network and transmission to a network. These operations can therefore be carried out at speed, overcoming the problems with known virtual machines and interpreters, which tend to operate slower than conventionally programmed devices. The virtual machine therefore lends itself particularly to applications relating to communications, such as payment terminal devices and other devices in which message pro-cessing and communication comprise a significant proportion of the operation of the device. In payment terminals, for example, a payment terminal including a virtual machine having the message processor means and protocol processor means can operate

satisfactorily speedwise. The virtual machine can be implemented on any hardware, BIOS/OS arrangement and therefore facilitates portability of programs.

Implementation of such a virtual machine on payment terminal devices of different brands enables operation of the payment terminal devices or brands to be altered merely by altering application commands generic to all brands. Each brand is seen by the application as the same virtual machine.

The virtual machine preferably also includes a function processor means arranged to control overall virtual machine action in response to operator or other external events, and also preferably includes function processor instructions which are arranged to provide directions for operation of the function processor means.

The function processor means is preferably a program mod-ule the specific function of which is to control and select general operations of the device not specially controlled by the message and protocol processor means.

The function processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The function processor instructions are preferably virtual instructions to be expressed only in the language defined by the function processor means- and thus never requiring translation to any real hardware processor.

In the preferred embodiment, the "application" will therefore comprise instructions for the message, protocol and function processor means. The instructions for the function processor means may include such prior art modules as a function event schedular and function selector.

Although the present invention is particularly applicable to application in payment terminals, it is not limited to such applications. The invention can be applied in any device where advantages are likely to be achieved for the arrangement and control of communications.

With the advent of the Internet and other extensive communications networks, it is believed that the operation of computers, such as PC's, will become more and more oriented towards acting as "servers" and/or "browsers". In other words, a major function of PC's connected to a network will be to operate either as a server, providing information and/or programs to the network for access by other parties, or as a "browser" for obtaining information/programs available on the network and operating on them. It is likely, in fact, that PC's will be asked to operate as both a'server and a browser. This operation will not merely be restricted to the Internet, but for any network, even Local Area Net-works.

The applicant also believes that many other classes of devices may be connected to a network. For example in the future a home video cassette recording machine could be connected to the Internet (along with other devices) allowing remote programming from a browser device. An example of the use of this would be a worker upon learning of a requirement to stay at the office late and miss a favourite show could access their home VCR from the office and pro-gram it.

Telephone calls will eventually be digital and most likely use the Internet as the digital network. Like the VCR, this does not mean all phones would need a qwerty keyboard and colour display. They will both represent other classes of Internet connected devices- not requiring the exact same configuration as PC's.

The present invention facilitates the production of a small, economical device which is particularly arranged to deal with communications, to build, compare and deconstruct message information. Such a device is novel maybe termed a Specialised Network Access Computer (SNAC). The applicants believe that a SNAC could emerge as a class of device allowing data entry and control through the Internet where a smaller, more economical device than a conventional PC is appropriate. In a preferred embodiment, the device is implemented utilising a virtual machine having a message processor and a protocol processor as discussed above. In the preferred embodiment, the software of the device can be considered to include three layers of virtual machine software (the HW drive layer, the Hardware Abstraction Layer, and the Virtual Machine Processor layer) and a software application. All layers other than the Virtual Machine Processor Layers are well established by prior art. A payment termi-nal can be used substantially without alteration as the hardware component of the device. A hardware abstraction layer (HAL) is a set of routines providing a common application program interface (API) to exercise the operating system, BIOS or hardware drivers.

HAL consists of routines to either (a) implement the functionality not provided by the underlying operating system, BIOS or hardware drivers, but needed for the common API, and (b) translation of parameters and adjustments of functionality required to adapted underlining OS, BIOS routines for the routines specified by the common API.

Such a SNAC can be applied in many different types of communication application over a network.

The present invention also facilitates the production of devices which incorporate a snac as a functional element of the device. Such devices could include both devices collecting information for transmission over a network such a pay telephones, particularly those equipped with smart card facility, or devices receiving information from a network such as the futuristic VCR or even washing machine.

Preferably, message instructions and protocol instructions may be developed on a convenient device such as a PC or general purpose computer, utilising a development tool in accordance with another aspect of the invention.

From a further aspect, the present invention provides a development tool for developing message instructions for providing directions for operation of a message processor means to be implemented in a virtual machine as discussed above, the development tool comprising a processing apparatus arranged to receive data input by a user to build message instructions for the message processor means.

The arrangement is preferably driven by a graphical user interface based program which provides various screens and fields for the user to input data relating to message instructions.

The message instructions are preferably subsequently converted to code and downloaded into the device which is to employ them with the virtual machine. From a further aspect the present invention provides a development tool for developing protocol instructions for directing operation of a protocol processor means to be implemented with the virtual machine as discussed above, the development tool comprising processing means arranged to receive data input by a user to build protocol instructions.

The arrangement is preferably a program which is arranged to build protocol instructions from the data input by the user. The program is preferably graphical user interface based and provides screens and fields to facilitate data input for the protocol instructions.

Protocol instructions and message instructions can therefore be built on a PC and downloaded to device where the virtual machine is to be implemented.

A tool has also preferably been provided for developing function processor instructions, along the lines of the tool for the protocol processor instructions and message protocol instructions.

Limited hardware provided by such a device as a payment terminal or other SNAC device does not lend itself to development and testing of applications programs. Although the finalised application must run on the hardware, to develop and test an application it is more convenient to be able to utilise a more user-friendly device, such as a PC or general purpose computer.

From a further aspect, the present invention provides a communications device including a virtual machine means including a protocol processor means arranged to organise communications to and from the device and protocol processor instruction means arranged to provide directions for operation of the-protocol processor means.

From a further aspect, the present invention provides means for emulating a virtual machine on a PC or other general purpose computer, the virtual machine comprising a message processing means and function processor as discussed above. The virtual machine is arranged to operate on the PC or other general purpose computer so that instructions developed for the machine can be tested.

Similar emulation is preferably provided for the protocol processor means.

Emulation can therefore be used to test payment terminal or other SNAC application programs.

The present invention yet further provides a method of programming a device for processing communications, comprising the steps of loading a processing means of the device with a virtual machine which includes a virtual function processor and function processor instructions for controlling operation of the device, and a virtual message processor which is arranged to be called by the function processor and which is arranged to carry out the task of assembling, disassembling and comparing messages, whereby when a message is required to be handled by the Add communications device the message processor is called to carry out the message handling task.

The method of programming preferably also includes the step of loading the processor means of the device with a protocol processor means arranged to organise communications to and from the device, and protocol processor instructions arranged to provide directions for operation of the protocol processor means.

The present invention yet further provides a computer memory storing instructions for controlling a computing device to implement a virtual machine means which includes a virtual function processor and function processor instructions for controlling operation of the device, and a virtual message processor which is arranged to be called by the function processor and which is arranged to carry out the task of assembling, disassembling and comparing messages, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task.

From yet a further aspect the present invention provides a computer readable memory storing code for implementing message processor instruction means arranged to provide directions for operation of a message processor in a virtual machine means, the message processor being arranged to process messages for communication to and/or from a device.

From yet a further aspect the present invention provides a computer readable memory storing code for implementing the virtual machine including a protocol processor means arranged to organise communications to and from a device.

From yet a further aspect the present invention provides a computer readable memory storing code for implementing protocol processor instructions arranged to provide directions for operation of a protocol processor means arranged to organise communications to and from a device.

From yet a further aspect the present invention provides a specialised network access computer, including a micro processor and a virtual machine means, the virtual machine means including instructions for running on a virtual micro processor and an interface enabling the micro processor to operate the virtual processor.

Preferably the specialised network access computer is a payment terminal or other type of "card computer" (being a computer which is arranged to process information from cards and/or communicate information to cards—cards being smart cards, magnetic cards or similar).

The interface between the actual processor and the virtual processor preferably includes a hardware abstraction layer (AJL) or the like which provides a common

Features and advantages of the present invention will become apparent from the following description of an embodiment thereof, by way of example only, with reference to the accompanying drawings, in which:

FIG. 1 is a schematic block diagram of a payment terminal in accordance with an embodiment of the present invention;

FIG. 2 is a schematic diagram of a control program architecture for the embodiment of FIG. 1;

FIG. 3 is a schematic flow diagram illustrating device operation which requires the operation of the message engine;

FIG. 4 is a schematic flow diagram illustrating an example of operation of the protocol engine;

FIG. 5 is a representation of a display (screen dump) available on a development tool for developing a program for a device in accordance with an embodiment of the present invention, illustrating development of a message instruction for an example message;

FIG. 6 is a screen dump of a further development tool display illustrating further detail of development of a message instruction;

FIG. 7 is a further screen dump of a development tool display illustrating further detail of development of a message instruction;

FIG. 8 is a screen dump of a further development tool display illustrating development of a further message instruction.

FIG. 9 is a screen dump of a further development tool display illustrating development of a protocol instruction;

FIG. 10 is a screen dump of a further development tool display illustrates further detail of development of a protocol instruction;

FIG. 11 is a schematic diagram showing a structural embodiment of the message instructions and description for the message processing means, and

FIG. 12A is a schematic diagram showing the structure of protocol instructions for an embodiment of the protocol processor means.

FIG. 12B is a representation of a display of a development tool for developing protocol instructions.

An embodiment of the invention will now be described particularly with reference to a payment terminal device. The invention is not limited to payment terminal devices and the following description is given as an illustrative example only. The invention can be employed in all devices concerned with communications over a network, such as a Specialised Network Access Device.

A payment terminal device in accordance with an embodiment of the-present invention is illustrated in FIG. 1.

The device hardware comprises a processing means which, in this embodiment includes a central pro-cessing unit 1 and a memory 2 for storing instructions and data.

The device further comprises a keyboard 3 for input; a card reader for inputting information from a card 5; a display 6; a printer 7, and a communications interface 8 for communication with an account acquirer.

Prior art devices generally have similar arrangements to that illustrated in FIG. 1. The number and type of peripherals to the CPU may vary, but the essential operation required by the prior art and the present invention are similar.

Such devices operate to facilitate remote payment transactions, and a general overview of operation is as follows:

(1) Information is taken from an account holder's (customer) card 5 via a card reader 4. Transaction information is input via the keyboard 3. The transaction information may include a money amount. The display 6 may prompt the user (merchant employee, customer) to input information (e.g., it may ask a merchant employee to input an amount) and may also display information as it is input. The keyboard 3 may also be used by the customer to input a code for the account, such as a PIN number.

(2) The CPU communicates the information via communications interface 8 with an account acquirer computer. The account acquirer computer may carry out a transaction (e.g., deduct money from the customers account and pay the merchants account) or may provide an "authorisation" that a transaction can be carried out. Information that an account transaction has taken place or that the account acquirer authorises a transaction to take place is transmitted to the communications interface 8 from the account acquirer computer. A display 6 may be provided to indicate that the transaction has occurred or may proceed.

(3) When the transaction is complete, a print out of transaction information may be provided from printer 7.

Prior art payment terminal devices are generally programmed in a conventional manner. That is, programming comprises a sequential set of operating instructions which are executed in sequence to carry out a remote payment transaction. This "sequential program" may be directly compiled onto the processor of the device so that the device is under direct program control or, as is more usual, an applications program in a conventional programming language may control operations through a BIOS/OS. Whatever conventional programming form is used, however, the device suffers from the problems which are discussed in the preamble of this specification. The programs are not portable between devices having different hardware or operating system architectures and it is necessary to write a program specifically for each type of device. Further, any amendments to the operation of the device must be programmed by a programmer having knowledge of that particular device and program arrangement.

FIG. 2 is a schematic block diagram illustrating architecture of a device in-accordance with an embodiment of the present invention.

The architecture comprises the hardware 100 the device, as illustrated and described in relation to FIG. 1. It also comprises the hardware drivers, known in the prior art, and including an existing BIOS/OS or hardware drivers, reference numeral 101 and also includes the Hardware Abstraction Layer Interface (HAL) 102. The HAL 102 and hardware drivers 101 form a layer of a virtual machine which also includes virtual machine processors 103.

The virtual machine 101, 102, 103 is arranged to emulate a hypothetical payment terminal. Application 104 15 controls the virtual machine 101, 102, 103 which in turn controls operation of the hardware 100. The virtual machine 101, 102, 103 can be adapted for many different hardware 100 arrangements (i.e. many different brands of payment terminal). Different arrangements of hardware 100 20 can therefore be controlled by the same application software 104.

The provision of Hardware Abstraction Layers and hardware drivers for virtual machines is known in the prior art and fully described in various publications. Each peripheral of the virtual machine is defined to be able to act in some manner on a standard set of commands. The HAL implements the best interpretation of each command on the actual peripheral present. For example a printer is defined to implement a "feed paper ready for tear off" instruction. On differing roll paper printers this requires feeding a different number of lines, on tractor feed printers this requires feed to the next perforation.

The virtual machine processors include a message processor 105 and a protocol processor 106, implemented in software code. The message processor is arranged to process messages communicated to or to be communicated from the payment terminal via the communications interface 8. The protocol processor is arranged to organise communications to and from the device, and to control and select the sequence of message processor operations in relation to messages received and transmitted. The message processor 105 and protocol processor 106 are implemented in native code of the payment terminal and therefore operate at relatively high speed. Because much of the "work" of the payment terminal is in building, comparing and deconstructing messages and processing communications, the operation of the device is relatively quick even though employing a virtual machine, 101, 102, 103.

The virtual machine processors 103 also comprise a function processor 107 the operation of which is to control and select general operations of the device not specially controlled by the message and protocol processors 105, 106. The function processor is also preferably implemented in the native code of the micro-processor of the hardware 100.

The application 104 includes protocol instructions 108, message instructions, 109, function support 110 and function instructions 111. The protocol instructions 106 govern operation of the protocol processor 106. The message instructions 109 provide directions for operation of the message processor 105. Function support 110 and function instructions 111 govern operation of the function processor 107. The application 104 and virtual machine 101, 102, 103 operate on data 112 input to the payment terminal to process it in accordance with the application 104.

In this example, the application include a set of "primitives" which are a series of symbolic commands which are executed by the device to control carrying out of a remote payment transaction. The appendix A to this specification lists primitives utilised by a preferred embodiment of the invention and gives descriptions of their respective functions. It will be appreciated, however, that a skilled person would be able to design their own primitives for carrying out remote payment transactions and the invention should therefore not be considered limited to use of the primitives listed in the appendix. It is in fact anticipated that users of the system may desired to created their own primitives and product documentation attached includes instruction for this procedure should it be desired.

Appendix A is in the form of a "HELP" file to be used with a product. The important information for the purpose of this description is the brief description of each "PRIMITIVE" and their function.

The primitives operate utilising the data 112. The data 112 may be data being input to the device, such as the customers account number, information which is fixed (strings) in the device e.g., a particular account acquirers identity.

The function processor 107 includes an event schedular and index as known in the prior art. In response to an event (e.g., swipe card) the event schedular operates via the index to look up a sequence of primitives 11 to be executed in response to that particular event.

In the preferred embodiment, the virtual machine processors 103 are constructed using C and the application is constructed using C++ or Java.

The device of this embodiment is event driven. When converting a device incorporation the SNAC hardware requirements to a SNAC by the provision of an appropriate HAL and virtual processors, and event driven structure can be added to a non-event driven underlying architecture through the HAL. This can be achieved through a software loop detecting events and generating an event call for any detected event.

The application 104 responds to the occurrence of an event to dictate subsequent operation of the de-vice. For example, when a card is swiped through card reader 4, the appropriate sequence of instructions from the application 104 will be implemented. The event driven structure allows the hardware drives 101 to have control during idle periods. When an input event occurs the application is called to process the input and then returns control to the hardware drives 101.

The application may be loaded on a remote payment terminal device with a pre-existing operating system. Where the operating system is event driven HW drivers 102 can operate as an interface layer without any problems. Where the pre-existing operating system (HW drivers) is not event driven, amendments must be made via the HAL to convert to an event driven structure.

Appendix B includes a description of a operation of the HAL 18 in accordance with an embodiment of the present invention, on a functional level. A skilled person would be able to develop an appropriate HAL structure for an existing device or a new device. The appendix B is in the form of a "HELP" file for a product. It merely describes an example of implementation of a HAL and adaptation of an existing devices existing BIOS.

FIG. 3 illustrates an example of an operation of the device, for one typical step in a remote payment transaction. The other steps in the remote payment transaction are carried out in a similar way. That is, they may require the operation of the message processor 105. They are event driven, such that the application 104 is called up to deal with any particular event after the event occurs, etc.

The operation schematically outlined in FIG. 3 is that of reading information from a customers card and storing information in fields for subsequent processing by the application 105. In overall operation of the device, the information from the card will be required to identify a user and enable access to the user account to cause a transaction or authorise a transaction.

FIG. 3A illustrates example information included on a magnetic stripe on a magnetic stripe card 5. The information includes track 1 information, track 2 information, track 3 information, the customer name, the PAN, the expiry date and End-Of-Form label. This information must be taken off the card and stored in appropriately labelled fields so that it can be accessed to enable processing of the transaction.

At step (1), on a card swipe of card 5 through reader 4, the card swipe event is detected by the HW drivers 101.

The HW drivers 101 causes a call back to an event table in HAL 102 for the peripheral card reader 4 which contains a series of names for routines to be performed on the occurrence of a particular event on the card reader 4. There are also event tables for the other device peripherals.

FIG. 3B is a schematic illustration of the event table for the card reader 4. Event "2" is for card swipe. In this example, there are three alternatives available for a card swipe event, labelled "1", "2" and "3". These labels may be dynamically updated in the event table, depending upon the particular stage of operation of the device.

Label "1" is for the routine "handle idle card". This is a routine which is instigated where no payment transaction routine has yet been instigated, i.e., this is "kicking off" operation.

Label "2" is the label for the "handle card" routine. This is where the payment terminal device is waiting for a card read event, e.g., where one has a device of the type which requires a money amount to be input before the card is read.

Label "3" is where the device may be at a stage in the operation where it does not require a card reader, i.e., the card is swiped in error. In this case, nothing happens and no routine is initiated.

Note that the above descriptions of the routines are not "primitives" but are merely general descriptions.

It will be appreciated that the event table may contain labels for any number of events to carry out operation of the device peripheral the card reader 4. Similarly the other event tables for the other peripherals will be configured with labels for various routines they are required to carry out, as will be appreciated by the skilled person. It is not necessary to go into detail detailing all the routines, as they will vary from device to device and will be a matter of choice of the skilled programmer, and the operator of the payment terminal device.

This event table driven structure is ideal. In a conventional terminal, where the terminal is executing sequential program instructions, for "handle card" routine the device will merely sit in a loop waiting for a card to swipe. With this architecture, however, the device does not have to sit in a loop waiting for a card swipe. It can leave the application program and return to the HW drivers 101 and in the mean-time the CPU 1 can be carrying out other jobs.

With the event label, the sequence of the apple-cation instructions for the particular routine is then looked up via an index from the application 104. The function processor 107 is then called up, step (3) to commence implementation of the instructions for card swipe. The function processor 103 then implements the instruction sequentially. The function processor 103 is a conventional interpreter, as will be understood by those skilled in the art, arranged to implement the high level primitives of the application 104 via HW drivers 101.

The first primitive requiring execution for the "handle card" routine in this example is the SAVE primitive, step (4). The first operation of the SAVE primitive is to call up the message processor 105. The message processor 105 is a series of several sub-routines implemented in the native code of the CPU 1, the specific operation of which is to construct, de-construct and compare messages in accordance with message instructions 109 from the application 104. The SAVE primitive will have associated with it a label indicating the particular message instruction 109 associated with this particular event. The function processor 107 fetches the message instruction 109 for this event and the message processor 105 then operates to load the data from the card into labelled fields (steps 5, 6 and 7) according to the message instructions.

Once the message processor **105** has loaded the information from the card into the appropriate fields, in accordance with the message instructions **109**, the SAVE function is completed and the device proceeds to carry out the next function in the sequence for "card swipe" fetched by the function processor **107**. Alternatively, the sequence of functions for "card swipe" may be completed and the device may wait for the next event before proceeding further.

There are a number of ways that the payment transaction could continue once the SAVE function has been carried out. For example, steps could be taken to create a display asking the customer to input a PIN. Again, such steps would be carried out by the function processor **105** implementing the instructions, which would include a function to call up the message processor **105** to build a "form" to display the request on the screen. Alternatively, the device could be controlled to take steps with regard to the information loaded into the fields by the card in accordance with the SAVE function. For example, it could compare a PAN number taken from the card with an equivalent PAN number stored in memory of the device to establish the identity of the account acquirer.

A skilled person will realise that a number of possibilities are available for continuing with the transaction, and would be able to formulate appropriate programming from this description and the following appendices.

As discussed, the message virtual processor means is directed by message instructions **109**.

FIG. **11** is a schematic diagram illustrating the structure of the message instruction means **109**. The message instruction means is in fact in the form of a set of "descriptions" of the messages. Each message usually comprises a plurality of fields **120**, and the message instruction means for each message contains a corresponding plurality of message instructions. One field may be the CUSTOMER NAME, for example. In the message instruction means, each field is associated with a number of message descriptors **121** which designate characteristic to be applied to the information in that field or to be expected of the information in that field. Operations which may be carried out on the data included in that field may also be included in the descriptors **121**. As illustrated in the drawing, the descriptors may include:

1. Data Location Identification. This will indicate either where the data is to be found and/or where data is to be put. In the current embodiment the data location information is contained in a two byte field descriptor (thus having 65535 different possible values) with value ranges allocated to
   1) 2000 strings
   2) literal numeric values from 0 to 32,000 in abbreviated form
   3) data field Ids where each ID is represented as an entry in a table, and each table may contain up to 256 fields.
2. Data Representation (i.e. Ascci, Binary, etc.).

This indicates what representation form the data is in and/or what it is to be converted to.

3. Format. This provides a description of the format that the data is in and/or is to be placed in.
4. Test Function. The index of a function processor set of instructions to determine if the current field is to be included or excluded at this time
5. Line & Column. Relative position for use in constructing messages for display or printing. These values are used to determine the quantity of space characters, and or new line characters that are required in the buffer.
6. Substitution list. A list of text representations to substitute for numeric values e.g., display the value "1" as "Monday" and "2" as "Wednesday".
7. Additional description options as required by the application or prove useful in future embodiments.

Each message instruction will therefore include a description of a field of message data, providing instruction for the virtual message processor means which enable it to carry out a number of tasks:

1. To compare a message with a message description to see if it is the correct required message.
2. To take a message of the correct description from a location and place it in an-other location.
3. To take a message and deconstruct it into various components and place the various components into other locations.
4. To take data and build a message in accordance with the message description and place the built message in a location.
5. Compare one message with another message.

Other functions may also be carried out by the message processor as required by the application. The message processor can manipulate data in any desired way in accordance with descriptions provided by the message instructions. Messages comprising data can therefore be billed, placed in locations, taken from locations, deconstructed with elements being placed in locations, etc. for subsequent operation on the data by the application. Any device which deals with significant amounts of messages in such form can therefore benefit from this arrangement.

Each message description is labelled so that it can be identified by the application, e.g. each message description may be numerically labelled.

A development tool for developing the application **104**, in particular the message and protocol instructions **108**, **109** comprises a graphical user interface based program which may be run on a PC or other general purpose computer. The program provides a graphical user interface based framework which enables message instructions to be built from data input by a programmer. Message instructions can subsequently be translated into code readable by the virtual machine **102**, **101**, **103** and downloaded into the application device. FIGS. **5**, **6**, and **7** are "screen dumps" which illustrate displays generated by the development tool for an example message instruction. In this case the message relates to data from a magnetic stripe of a customers card. The message instructions direct the message processor **105** to take the fields of the message and place them in known locations in accordance with the instructions. Such a message instruction may be called up in response to the SAVE primitive, in the event of a card read. Data from the magnetic stripe of the card would be stored away in the appropriate locations in accordance with the instructions, for subsequent processing.

Each message is provided with a message name **30**, in this case "TrackData". This message name identifier can be used to call up this particular set of message instructions in the development tool. An alternative numeric identifier is generated for use by the virtual processor. This numeric identifier may also be displayed by the development tool. Each message is made up of a number of message "fields" **120**. In this particular example, there are seven fields, being "Track1", "Track2", "Track3", "CustName", "PAN", "ExpD" and "End-Of-Form". Each of the seven field is converted to a message instruction for use by the virtual message processor. This is the information which is typically found on any magnetic stripe card. The message instructions in accordance with this embodiment direct the message processor to process these elements. Each field is associated with descriptors which provide further instructions for the handling of that element. FIG. **6** illustrates a display **33**

which enables a programmer to provide message descriptors to CustName element.

Each field 120 has a "format" descriptor 34.

There is an instruction as to the Data Representation ("Type") reference numeral 35. In the illustrated embodiment there are four types, AscII, Hex, Binary and BCD. There is also a logical operation instruction (option test), reference numeral 36. This logic instruction can be used to determine whether or not the message processor will process this element at all, for example, i.e., it will only include the CustName element in the message when the logic function equals "True". Other instructions designate the data source, reference numeral 37, in this case a field, and the field label, reference numeral 39. The format 34 is labelled with a name, in this case, "Tracks". There are further instructions which dictate the format Tracks to be applied to CustName. FIG. 7 shows a display which illustrates the instructions for the format "Tracks".

The message processor is responsive to all the message instructions to load the data from the magnetic stripe card into the appropriate fields with the appropriate formats in accordance with all the rules designated in the instructions.

This embodiment of the present invention includes another class of message instruction means, known as a "Form". Instead of a Data Representation as a message descriptor, a Form includes description of a Location of the data field in the Form. FIG. 8 is a display provided by a development tool enabling the programmer to prepare message instructions for a Form message. On the left hand side of the display a panel 70 illustrates Form layout. The fields in the Form include MerName, Address Line 1, etc. The location of these fields can be moved within the panel 70. The location in the panel is provided as a descriptor and for the message instruction. The Form type of message instruction controls displays, reports, print-outs, and the like. The type of Form is given by the instruction designated by reference numeral 71, in the example illustrated in FIG. 8 being a print-out. The message processor takes the fields from known memory locations or other locations and enters them in the locations enabling the Form described by the Form instruction to be produced.

As discussed previously, another major function of a SNAC device is communications. For example, it is necessary for the majority of remote payment transactions for communications to be able to occur between an account acquirer location, in order to enable access to an account, and the remote payment device. Communication with a data carrier, such as a smart card device may also be required.

The protocol processor 106 is arranged to organise communications, in accordance with directions from the protocol processor instructions 108. Referring to FIG. 4, in a typical remote payment transaction, after a card has been swiped, a PIN number has been input and a charge amount has been input, information then needs to be communicated to an external computer, at the account acquirers, in order to enable further processing of the transaction. After an event such as a communications message arriving, therefore, HAL 102 detects the event (step 1) and activates the protocol processor (step 2), FIG. 4. The protocol instruction 108 for the event is rolled up (step 3). The protocol processor 108 implements the protocol instructions for that event, (step (4)).

The protocol instructions are divided into "sections" 130, "lines" 131 and "protocol commands" 132, as illustrated in FIG. 12A. FIG. 12B illustrates how an instruction is displayed on a development tool for protocol instructions. Protocol instructions describe message flow both from and to the device. The top line specifies outgoing messages and the other lines display possible incoming results. A protocol consists of lines and sections. At the start of each section is a line 1 (optional for the first section) which describes the outgoing message. There are a number of protocol commands, and these include:

1. Protocol—Run a sub protocol
2. Message—Send a message or handle an incoming message using the virtual message processor means
3. Retry—re execute the steps of protocol from and indicated point
4. End—End of the protocol
5. Exit—Stop the protocol from an intermediate point
6. Timeout( )—Specify the a delay after which the protocol should automatically jump to the point at which the timeout instruction is placed.
7. Control—Specifies a control character to be send or received.
8. Function—Execute a virtual function processor function

Protocol instructions are organised in lines and sections. In each section Line 1 indicates the information to be send by the SNAC device and subsequent lines indicate actions to be taken in response to the alternate possible events which may occur in reply. The first instruction on each of these subsequent lines is used to identify the response. Control( ), Message( ), Function and timeout( ) may all be used to identify responses as follows.

1) When the time specified by a timeout instruction elapses then the line commencing with the timeout will be selected.

2) When data is received it will be sequentially compared to a lines commencing with Control( ) Message( ) or Function to see if the data matches the control character, matches the message of causes the test contained in the function to evaluate to true.

FIGS. 9 and 10 illustrate displays of a development tool for protocol instructions for the protocol "General" which is the Protocol Name (reference numeral 42). Instructions are presented as a screen dump in the form of a table 43, which can be accessed by a programmer if he wishes to alter the protocol.

Protocols are arranged to control message flow both from and to the target device (e.g., account acquirer computer). The top line of the display panel 44 specifies outgoing messages and the other lines display possible incoming results.

A particular protocol is able to call up other protocols "nested" within it and is also able to call up the message engine to deal with messages.

Referring to FIG. 10, the top line of panel 44 specifies the outgoing message. The first operation of protocol "General" is to call up and carry out a further protocol, "Reversal". FIG. 11 illustrates instructions for the protocol Reversal, reference numeral 45. Reversal operates to call up the message engine to construct message number 0400 and this message is then sent to the target device.

The either

1) Message number 0410 should then be received back from the target device and the message processor will be called up to deal with that data, which involves the message processor comparing the incoming message against the description specified by the message instruction means and storing the data if a match occurs. Or

2) A timeout of 100 tenths of a second elapses.

Then the protocol is ended and re-turned to the protocol General, which causes a further message, 0100 to be formulated and sent out.

Then either

1) A message matching 0110 will be received or

2) A message matching 820 will be received or

3) neither 1 or 2 will occur for the timeout( ) period, in this case specified as 000 tenths of a second.

If the message 0110 should then be received from the target device and compared by the message engine, then another protocol "adjustments" will then be carried out. The protocol would then end.

If the message 820 should be received from the target device, which can be dealt with by the message engine and compared with the instructions from the message instruction means. The "Retry" instruction will then be executed causing the virtual protocol processor to move execution back to the sending of the (0100) message. The retry count of zero indicates this loop would continue whilst 820 messages are received.

If the Timeout occurs, then the retry(5) would be applied causing the protocol processor to move execution back to the Send 0100 message. This loop would occur up to five times as indicated by the retry(5). After the fifth time execution would move to the next section causing the protocol to End.

More details of operation and build up of messages and protocols are given in the appendix A.

The device in accordance with the present invention, for example a payment terminal, may be implemented in GAVA by defining a class library payment terminals. This class library would contain calls to all the functions of how HAL and preferably the message and protocol engines. Similarly, a specialised network access computer or card computer could be implemented in GAVA.

Please note that the arrangement of the present invention can be used to deal with any payment transaction device, including one which deals with smart cards.

The present invention can also be used to implement a specialised network access device, which may use similar hardware to that provided for a payment terminal.

In the attached Appendix A, the term "CardScript" is the name the applicants have given to programming required to implement this embodiment of the invention.

It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the invention as shown in the specific embodiments without departing from the spirit or scope of the invention as broadly described. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.

APPENDIX A

Contents.

Introduction

Introduction

Help for CardScript Scribe

The Scribe program assist in the design of stored information & programs for EFTPOS terminals, PINpads, Electronic Cash Reqisters and other small computer systems.

Writing A Program

For help on writing a CardScript, program, rather than operation of the Scribe tool, see

Writing A CardScript Program

A CardScript program is more similar to a Windows RAD tool program than a conventional C Language or Assember program.

The "target" device has several keys, one or more card readers, and usually one or more communications ports. Defining a program consists of attaching actions to these events, or the special events of terminal power on and terminal idle.

CardScript programs—as all other program—manipulate data. Data is defined in a Data Dictionary. Unlike normal programs, it is normal to write many CardScript programs using the same Data Dictionary. Standard Data Dictionaries are available from CardSoft for EFTPOS and several other application types. It is recommended to write initial applications based on one of these standard dictionaries. Once the program is experienced, the Data Dictionary for an Application may be modified, see

Configure Data Dictionary

Data Dictionary Usage

The Data Dictionary represents the list of all "variables" or information values used in the target device. These "variables" are in formation which may change over time, or be different from device to device.

Information which is fixed for all devices usually is defined by strings. All information to be included in displays, receipts, messages etc, comes from either the Data Dictionary or from STRINGS.

Data Dictionary fields may have an initial value set from the Initial Data Tables

Structure

Tables

The Data Dictionary is divided into tables. Each record displayed in Configure Data

Dictionary describes one table. Fields are placed by selecting add and clicking on the Panel.

Field Attributes

Double Clicking on any field reveals and allows viewing and/or editing of Data Dictionary Field Attributes.

Field Order

Layouts are stored indexing fields by table#1field#. This means existing scripts will behave strangely if the Data Dictionary is changes the number Of referenced fields.

For example if "Merchant Name" is table 3/field 2 and "Address" is table 31field 3. Then deleting field table3/field1 will make any prior references "Merchant Name" now reference "Address". This can be remedied by inserting a dummy table3/field 1 as a placeholder. Generally this problem does not arise since new dictionaries are not to be used for old applications, and existing dictionaries are usually only extend. In the rare event that a dictionary used by existing applications is to have fields deleted, it recommended to rename them to "dummy" or "unused".

Be careful since any existing data in the files will be rearranged when retrieved, it will simply be move from the record into the fields in the order listed at the time. New fields added in graphic display mode are always added at the end

Reserved Settings

see Reserved Data Dictionary Settings

see also

Data Dictionary Field Attributes

The field attributes which may be set are as follows

Type

Type refers to the format in which data is held. "X-Ref' is a special value used to indicate that another table will be referenced at run time and thus must be included in the build.

Binary Data Fields

Binary. either 1 or 2 bytes in length for Integer values in calculations, longer fields hold bit fields or keys. 250 bytes is the maximum permissible number of bytes

Maximum Integer values

Depending on the number of bytes used to represent the Binary number, the following values are possible

1 Byte	0 ...	255
2 Byte	0 ...	65,535
3 Byte	0 ...	16,777,215
4 Byte	0 ...	4,294,967,295

Text—up to 250 bytes

BCD up to 250 bytes

Date/Time (2 Bytes for Dates, 2 or 3 bytes for Time) see Date & Time Fields

Amount—10 bytes, internal format is target device dependent

Packed Amount—not currently used

X-Ref—Advanced use only

Flags

0=Field is fixed and never reset

1=Reserved for future use

2=Reserved—used with deleted fields

3=Field is reset when terminal is loaded

4=Field is reset at power on

5=Field is reset by idle function

Bytes

The leright of stored data in bytes

Length

Caution: When you create new data dictionary fields, make sure their leright is not zero if you want to use them, or they will be invisible.

The number of characters allocated to display the field as text

Name

The name of the field for display on receipts etc.

Table

The "refer" Initial Data File from which the field initial value is extracted. Blank if the field is extracted from the default file.

Table ldx

When "Table" is non-blank, "Table ldx"specifies the "refer" of the Initial Data Field in the default file used to indicate the record number in "Table" from which data is to be extracted. In other words the join field between the default table and the joined table.

Field

The "refer" of the Initial Data Field from which the initial value of the field is to be obtained.

Creating a New Application

As suggested above to create any application, it is recommended to copy a "template" application. Simply copy the entire template directory.

To then work with the new directory, select File/Installation and edit the data directory. (Don't forget the trailing I) Is the recommended to exit Scribe and restart. Scribe and restart.

Console/Display

The display console used with CardScript is quite sophisticated. see

The Console

CardScript can be used in a variety of devices, some of which may not support all the features described here.

Features

The CardScript Console has a number of sophisticated features

Hot Keys

Keys used to launch only one action, where the action is part of the application, are known as hot keys. Typically the action may be activated only when the terminal is idle. For further information see the "KeyBd" primitive.

In an EFTPOS application, on a terminal with Keyboard Buttons available for allocation, Hot Keys will normally be allocated to such functions as "Sale", "Adjustment", "End of Day" etc.

Hot keys normally would have their label printed on the keyboard, or on the physical button.

Multiple Field Input

On any Layout displayed on the console, several field may be selected for input. The OK key steps from one input to the next. Any soft key terminates all input.

Scrolling

The display may be scrolled, permitting a larger virtual display than the physical display. Scrolling is performed automatically by the driver in the target device. All that is needed to enable scrolling is to tell the scrolling driver what keys on the keyboard perform scrolling. The keys used to scroll are set by the

Console Primitive

Console(Command,Parameter)

The command determines which of the following console options is set.

Command 1—Set Scroll Keys

The Parameter is a string of four hex values, in order—key-left,key-right,key-up,key-down

The keyvalues specified are assigned to the scrolling engine within the target device. Note scrolling my not function on all CardScript "targets" and the size of the scrollable area may vary.

Command 2—Set Keymap

The parameter is a Key board map. This command is now the preferred method for setting the keyboard map. The old Keymap primitive will be obsolete in a future version. See

KeyMap

Structure of Keymaps

The"field" or String used as a keymap must be a list of 4digit-hex blocks, the first two digits of each block representing the hex code of the key to be mapped and the next two digits representing the hex value to map be returned. Usually used from the startup function, using a field from the terminal groups record.

Note that the following key codes have special meaning.

08

Represents a backspace or 'CLR' code.

OA

Represents an 'OK or 'Enter

1B

Represents a cancel.

30 through 39

Represent the digits "0123456789"

Command 3—Keybd

The parameter is evaluated to zero, or non zero.

Upon entering idle state, the action of the keyboard is determined by the last KeyBd command. The keyboard (except cancel) will be ignored if off was specified.

This command replaces the old keybd primitive, which will be obsolete in a future version.

Command 4—Invalid Entry

The parameter is the text message to be displayed.

This command is designed to be called from an input validation function. Calling this command indicates the input is invalid, the text specialised in the parameter should be display to indicate the error.

Soft Keys

Some keyboard buttons on a target device may be used as soft keys. As opposed to Hot Keys these are buttons which may be used to initiate different actions, depending on the display present when they are pressed.

Since the principle of Soft Keys is to use the same buttons for different actions, displays must in some way indicate to the user the operation of each currently active soft key.

Soft Key Button Sets

Target devices may allocate certain buttons on the keyboard for use as soft keys. Button sets are numbered from zero. If a specified set is not available, then set zero is used. By convention:—

Set 0=keys 'O' thru '9' (the numeric keys)

Set 1 are dedicated soft keys, usually positioned directly adjacent to the display, in order that the display may be easily used to indicate their function.

Set 3 is the new standard for dedicated soft keys—hex values 81,82 . . . AO.

Set 3 will normally be requested on forms where numeric/text input is also required.

Set 4 is the same as set 3, but allowing use of the numeric keys if no dedicated soft keys are available. Set 4 should not be specified on screens where numeric input is also available, since this may cause a conflict.

Soft Key Action Groups

These are the groups of actions that may be offered at any given time.

In Layouts/Forms, a soft key action set may be selected for any display. Individual functions may be assigned to an action group from the Function/General Purpose Functions, in the Function activation section.

Note Action group 0 is used to indicate a function is NOT part of any group

Correlation of Actions to Buttons

If a display allows soft key Button Set 0, (Keys '0', '1', '2', '3' etc) and action set "I" then when the '2' key is pressed, then soft key group '1', An Group '2' (Key '2' minus the first key in the action set equals 2), if it exists, will be activated.

The Keyboard

To control the features available, and make best use of your terminal,you can recode the keys on your keyboard using the keymap primitive. This allows you to customise the target device to allow portable and easy to operate applications. See

Keyboard Codes

The Keyboard codes are designed to accommodate a wide variety of keyboard configurations. At any given time each key (or button) will act as one of the following key types

1 Control/Data Entry Control

2Data Entry

3Soft/Hot key function activation

Control/Data Entry Control

Some keys are required for control. Control keys should not be used for any other purpose than control, otherwise the user interface will incredibly confusing.

Minimum Requirements

All CardScript drivers should provide these key codes without any mapping required.

08 Backspace or Cir

1B Cancel

OA OK or Enter

1A Fn—For general Function selection, and for double (or triple) zero.

Additional Options

OD or complete form (combined with tab as an alternative to OA)

09 Tab (move to next field—does not complete form)

OB Vertical Tab—Used as back tab or move to prior field.

1 1 (XON)DCI—Used as cursor left

12 DC2—Used as cursor right

13 DC3—Dedicated double zero

14 DC4—Dedicated Function Key (combined with DC3 as an alternative to 1 A).

Data Entry

Three levels of data entry may be available at any one time. Text, Hex,Alpha. The bios can automatically determine the available level and act accordingly.

Minimum Requirements

30.39 (Numerics)

Additional Options

A B C D E F (allowing Hex data entry)

Full 'query' keyboard

Soft & Hot keys

Soft keys previously were recommended to be 'a''b–c' etc Now the are recommended to be hex 81 82 83 etc up to a maximum of AO allowing up to 32 dedicated Soft keys. The change in recommended values is to allow for terminals with full alphabetic keyboards.

Hot Keys (When/Additional to soft keys) should be allocated from Al . . . DO

Program Portability

Portable Programs

CardScript allows the writing of totally portable programs, it is also possible to write programs that are not very portable. Any CardScript program will "execute" on any CardScript enabled target, however the result could be of no use on the target if special hardware characteristics are required for practical operation of the program. CardScript provides a mechanism for avoiding the traps and keeping programs portable whilst still taking advantage of special hardware when available.

Keyboard Traps

The key map primitive represents a trap in that this function should never be used with a literal string, or your program won't be portable.

Processing Cards

Magnetic Cards

Automatic Processing

Automatic Magnetic Card Processing, from

Upon Card read, data from the card is placed in the Receive buffer. The format in the buffer is

Track 1 (I terminated)

Track 2 (I terminated)

Track 3 (I terminated)

Customer Name (I terminated)

PAN (I terminated)

Expiry Date (6 bytes ASCII)

If the read occurred at terminal idle, the Calculation Result is set to zero and the system event Magnetic Card Read is generated. After executing any function for the Magnetic Card Read Event, if the Calculation Result is non zero this value is used to select a function for further processing of the specific card type.

Automatic Magnetic Card Processing

Upon card swipe, the data dictionary fields Transaction/Track2 tru Transaction/Customer Name are filled with the card details. These are data dictionary fields Tablel/Field2 thru Tablel/Field7. see Reserved Data Dictionary Settings for details.

Table 5 is then scanned to find a column matching the PAN of the swiped card. If a column in table 5 is found then the tables 3 & 4 have their columns set according to the entries for Issuer & Acquirer in Table 5.

Table 5 is set during the build to indicate the appropriate action should a card be swiped at idle. If the terminal was idle at the card swipe this function is now executed.

Typical Processing.

For standard processing, create a function as follows

store (O,CardMsg)

if ColFind(Pan,PanLow,PanHigh)

ColSelect(Issuer, IssuTbl)

ColSelect(Aquirer,AcqTbi)

Exit(O,CardFunc)

Smart Cards

Two primitives are available for controlling Smart Cards.

Card(Command,Field/Value)

1A command of 1 is used to read the smart card status into the field "FieldNalue". Using a value for "FieldNalue" does achieves nothing.

2A command of 2 is used to control the power to the card.

If "FieldNalue" is 1, the card is powered on, if "FieldNalue" is 0 the card is powered off.

3Select. A command of 3 is used to select which smart card reader(or plug in is currently selected. By convention, 1 is the user card(or if only one reader is present, this reader), 2 is the separate merchant card slot or the plug in, where present. "FieldNalue" contains the card number to be selected.

4A command of 4 is used to read the Smart Card Type Code

5A code of 5 performs a logical test on smart card type. If the field/value supplied matches the Smart Card Type Code of the current code, the logical true flag is set. This command is designed to be used in an "if" test

6Code 6 reads the CardEntryMode into the specified FieldNalue. See CardEntryMode 7Set Card entry mode to the value specified in FieldNalue

see Also

TPDU(Command, SendMsg,RxMsg,Status)

The TPDU primitive is used to send a command to the card.

If the TxMsg is present this data is also sent to the card.

If the RxMsg is present then a response is expected from the card and is stored in the RxMsg buffer.

Command

This if the actual 5 bytes TPDU to be send to the card

SendMsg

This optional parameter specifies the message used to build the data send to the card.

RxMsg

This optional parameter specifies message used to store any data returned by the card in response to the TPDU.

Status

This mandatory message specifies the location of status variables to record the status of the TPDU operation.

The TPDU primitive with Synchronous (Memory Cards) 416 Cards

Drivers for 416 Cards support the following TPDU Commands

ReadBytes

WriteBytes

EraseBytes

Present Key

see Commands for Memory Cards

The present Key uses the length indicator to select either the CardSecret Code (2 bytes) or the application erase secret code (4 bytes)

Answer to Reset, & Card Type

The 416 has a card type code of 4 and an answer to reset of

3Bh 00 00 00 00 00

Commands For Memory Cards

ReadBytes

CL (any)

INS BO

ADDR XXXX (Byte Address an Card)

LN LL Number of bytes to read.

WfiteBytes

CL (any)

INS DO

ADDR XXXX (Byte Address an Card)

LN LL Number of bytes to write.

EraseBytes

CL (any)

INS DE

ADDR XXXX (Byte Address on Card)

LN LL Number of bytes to erase.

PresentKey

CL (any)

INS 20

ADDR XXXX (Byte Address on Card)

LN LL Lenght of Key.

Other Smart Card Commands

For selecting the smart card reader, and control of the reader.

For sending information to the card, and receiving information from the card.

Smart Card Type Codes

1 Async ISO type Card 2416

Asynchronous SCHLUMBERGER type EE2K

Asynchronous SCHLUMBERGER type EE4K.

Asynchronous SCHILUMBERGER type EE16K.

Asynchronous type GPM256

Asynchronous generic type 12C BUS

Asynchronous type GFM2K

9 synchronous type GFIV14K

For coding of smart card types.

Running A CardScript Program

To run the program on the PC simulator, see

PC Simulator

There is a implementation of the CardScript Virtual machine available on the PC that not only runs your program, it also emulates the keyboard layout and other controls of any target machine.

To run the simulator—select "Build/Run Simulation of Build"

Stored Information—Data Tables

For Information on setting & changing values in the Data Tables See—

Tables Menu

CardScript includes all tools for maintaining data tables to control the set up and distribution of Data Tables required for any application.

The System Data Table

The system data table has a fixed format identical in all systems. This table contains general information and current setting for use within the scribe program.

See—

System Table Settings

Settings in the system table are used to both miscellaneous settings in the script, and options for viewing the script.

Loader Settings

Terminal For Mask Load

Not used by scribe

Default Prompt (or String) Table

The string table displayed within Scribe. Multiple string tables may be used to support multi language applications.

Peripherals

Description the peripherals of a the target system here and displays and receipts will in scribe will show guides to assist

in design. These setting have no affect on program execution in target devices and may be changed at any time.

Reserved Functions

Idle State

Set this pointer to indicate a function to be executed each time the "target" becomes idle.

Abort

Normally left at <none> since applications may vary default options during execution.

Initial

This function is executed at "target" power on.

Processor

Previously used to indicate the byte order used in the "target". It is now recommended to use "Low-High (INTEL)" for all systems.

Configurable Data Tables

The data tables used by CardScript can have their names, field names, layout and even the number of tables used altered according to the current system set up.

Configure Initial Data

Initial Data Usage

The purpose of initial data tables is to provide a database of information for initial values for the data dictionary loaded into the target device.

Configuration

Structure

Each record in the configuration describes one table. Fields are placed on the Panel and dragged to the appropriate position.

Field Attributes

Double Clicking on any field reveals and allows viewing and/or editing of Initial Data Field Attributes.

Field Order

Clicking on "Graphic Display" toggles between the standard graphic view of the fields and a simple ordered list of the fields. In the ordered list mode fields may be dragged to rearrange the field order.

Be careful since any existing data in the files will be rearranged when retrieved, it will simply be move from the record into the fields in the order listed at the time. New fields added in graphic display mode are always added at the end.

Reserved Initial DataBase Settings

Certain fields must be present in the for the build process.

File Usage

File 1—"Terminals". The name may be changed however this file is used to initialise individual target devices with the optional "NetMgr" module. No other special usage at present

File 2 "Groups" —no special considerations

File 3 "Issuers" no special considerations

File 4 "Aquirers" no special considerations

File 5 "Card Ranges"—must be used as card ranges, and must have fields "lo" "hi" and "len"

File 6 "Products" no special considerations

File 7 "Region Settings" no special considerations

File 8 "Issuer Sets" no special considerations

File 9 "Card Sets" must contain the fields "cards" as an index into card ranges

see also

Initial Data Field Attributes

Type

Flags

Current usage

0=Place label to Left

1=Place lable above

Repeat

The number of times the field is to appear on the form

X-Pos

The current value of X-position of the field on the form. Usually modified by dragging the field.

Y-pos

The current value of Y-position of the field on the form. Usually modified by dragging the field.

Label

The label to appear for the field on screen.

Refer

The "refer' label used to access the field when building the initial data dictionary

Display (Type=Text only)

The number of characters to be displayed on screen. 0 (zero) for default.

Size

Functions

For information on defining functions in your application see—

Functions Menu

see also Function Primitives

For Describing any function within the "target" to the system, or in program terms for writing scripts see

General Purpose Functions

Use this selection for describing functions

Label

The function name

Description

A brief description of the function. The function can be located by description

Action

A window to the function actions. Double click on this window to see or edit the full Function Action. see also Function Primitives & Function Primitive Categories. see

Function Action

Double clicking on a function action block brings a panel into view for editing the function actions.

Adding Actions

Select the appropriate action from the alphabetical list beside the add option, select add and then click on the panel at the appropriate position. Clicking over an existing action will insert the new action before the existing action

Deleting, Actions

Select delete and click on the action. Take care to deselect delete before clicking on other actions.

Editing Actions

Double Click on any action to activate the edit dialog box.

Function Index

Shown for reference purposes. Cannot be changed.

Strings

See your driver information. Currently this information is not used by most drivers.

Function Activation

Specifies when this function will be executed. see

Starting A Function

Function Number

Each function may be assigned a number. The operator may then enter the number and the program easily select from the list using the Function# primitive.

Hot Key Code

Each function may be assigned a hot key code. Enter a non-zero code in HEX and if a key with this code is pressed at idle, or any other time hot keys are activated, the function will be activated. Note that the Cancel Key is regarded as system event.

System Events

System Events are similar to hot keys, only instead of keys being pressed (Note that the Cancel key, IS a system

event), other actions on the target device are involved. For each target machine a list of System events is maintained, but these should always include the standard events. Only one function may be assigned to any System Event.

See

Standard Event Codes

Keyboard.

Keys on the keyboard with a value less than 128 (Ox8O hexadecimal) generate an event code with the value of the key.

Other event Codes

0=Reserved

1 System becomes Idle

2Cancel Key Pressed

3System Power On

4Numberic Entry

5Smart Card Insertion

6Smart Card Extraction

7Magnetic Card Swiped

8Checksum Error Detected on Batch

9Checksum Error Detected on Data

Card Set

Select a card set. When any card belonging to this set is swiped at idle, the function will be activated.

Usage Flags

Operator Function—The operator of the target device selects the function

Library Function—The function is an internal "subroutine" Not Used- The function is not used

For adding actions to functions which may be varied by issuer or by acquirer see

Transaction Function Input

Transaction Function Approval

Function Primitive Categories.

Function script is a sequence of calls to system and user primitives. For information ol primitives available see

Function Primitives

Assignment Primitive

Field1:=String/FfeW

Set field1 to the string/field2.

=> (goes to) primitive

=> field

The value of the last logical or other operation using the "calculation result" is stored in field

specified

eg

Account==000

=> zAccount

Would set the field zAccount to 1 if Account was zero, other size zAccount would be zero.

=> result (goes to) primitive

=> field

The value of the last logical or other operation using the "calculation result" is stored in field specified

Account==000

=> zAccount

Would set the field zAccount to 1 if Account was zero, other size zAccount would be zero. see also

Calculation Result.

Calculations generate a "Calculation Result". Think of this value as the value you would see on the display of a calculator if the calculation was performed on a calculator.

Math Primitives

Field1+=Number/Field2

Field1−=Number/Field2

Field1*=Number/Field2

Field1/=Number/Field2

Field1 is modified by thefield2/number.

Relational Primitives

Fieldl1value1<Field/value2

Fieldlvalue1>Field/value2

Fieldlvalue1==Field/value2

The two fields or values are compared. If one field is text and the other numeric then the return value will always be false.

< > != >= <=

For not equals (whether thought of as < > or !=), greater than or equals (>=) and less than or equals (<=) use the opposite case. With the WHILE PRIMITIVE and REPEAT UNTIL primitive then use the NOT option. With the IF PRIMITIVE use the ELSE clause.

Abort Primitive

abort

This primitive causes the target device to stop all functions and become idle

Alarm Primitive

Alarm (noise type)

Makes the sound specified

1 error alarm

2bip type noise

3most severe alarm

Bit Manipulation

Bit Numbering

All binary fields can be accessed as a number of bits where the number of bits no.of.bytes*8

The MSB of each byte has the highest bit number and the LSB the lowest bit number. Note ISO bitmaps do NOT follow this rule, but these do not need bit manipulation by the application.

This result in the LSB of the last byte being bit 0 (zero) and the MSB of the first byte being no.of.bytes*8 −1. Eg for two bytes 15.

Bitcount Primitive

bitcount(field,start,end,bitvalue)

start & end

These are both bit numbers. see bitnumbering. Direction of counting is from start to end, either may be the larger number.

bitvalue

0 indicates count zeros

1 Indicates count ones

2 Indicates counts zeros and stop at the first non zero bit

3 Indicates count ones and stop at the first bit no set to one.

Notes

The number of sequential bits of the value "bitvalue" starting from bit "start" and working towards "end" is counted.

If the result is non-zero the logical true status is set, otherwise the logical false value is set, allowing "if" type tests of the result

The count is stored as the "working value" allowing storage via the "->" (goes to) primitive. see -> (goes to) primitive

Setbits Primitive

Setbits(field,startbit,endbit,value)

Bits number "start bit" thru "endbit" are set to the value "value". No all values are extracted from the least significant bits of value. e.g. For a 1 bit field, all values are considered either 1 or 0. (Odd numbers are 1)

Batch Primitive

Batch (Operation)

Operation 0=reset to first txn in batch

Operation 1=find & restore next txn

Operation 2=delete current transaction

Operation 3=delete all transactions

CardRead Primitive

Card (string, field form, default)

This primitive is identical in operation to the show primitive, with three extra facilities

1Input is terminated by either the introduction of a smart card, or the swiping of a magnetic card.

2Data from a magnetic card read is stored in the reserved fields

3If the (card entry mode) is non zero, this primitive does nothing. This allows logic to read a card only if the card is not already read.

See also

Example

A function requiring input of both "Tip Amount" and "Cash Out Amount" can -input both using the same field.

Create a form as follows. Edit the PFIELD to indicate an input field.

PSTRING Name: Form

PFIELD

Create a function

Show(Tip, TipAmt, Form, 0)

Show(Cash, CashAmt, Form, 0)

Where "Tip" and "Cash" are strings. On the first call to show the display will prompt "Tip" and accept input into the TipAmt field. On the second call to Show the display will prompt "Cash" and accept input into the CashAmt field.

Show (string, field jorm, default)

Action

This primitive is specialised for displaying input forms. Two parameters (string and field) substitute with PSTRING and PFIELD in forms, allowing the same form to be used for multiple inputs.

String

String to replace the PSTRING field on the form. <none> if unused.

Field

Field to replace the PFIELD field on the form.

Form

The Form may be selected from the list box—or alternatively by selecting Field-> Value[X] taken from a field in the data base.

Default

A value of one (1) if the existing value of the field is to be displayed as a default, otherwise 0.

Reserved Data Dictionary Settings

The driver in the "target" makes direct use of some fields in the data dictionary. Using these table#/field# settings for other use will have strange results and is not recommended.

Table 1 (Transaction Table)

Fields in this table may be initialised to default values only. The first fields in the transaction table are reserved for (in order)

1ROCNUM
2Track 2
3Track 1
4Track 3
5PAN
6Expiry Date
7Customer Name
8Protocol Status
9Card Entry Mode

Table 2 (Totals Table)

No reserved settings, however a fixed ten copies are available. Initialisation of fields to default values only.

Table 3 (Terminal Table)

This table is the basis of the build of terminal groups, and may be initialised from the Initial Data Table. There is always only one record in the table.

Table 4 (Issuer Table)

One record per issuer, with the current record selected automatically when a card is swiped.

Table 5 (Acquirer Table)

One record per acquirer, with the current record selected automatically when a card is swiped.

Table 6 (Card Table)

Fixed layout, Dictionary specification currently ignored.

Coffind Primitive

ColFind(value,LowField,HighField)

Both LowField and HighField must be in the same table. This table is scanned for a column with the value 'value' between the two fields. The primitive is normally used to locate the CardTable Column for a card. The result variable is set to 0 if no match is found, or 1 if a match is found.

ColSelect Primitive

ColSelect(Column, Table, Reset)

Selects the relevant column of the table indicated.

Column

The column to use. The transaction table has only one column, the totals table has ten. The other tables (issuers, acquirers etc have one column for each record in the corresponding initialisation data base

Table

For comparability, 0 (zero) selects the totals table. The tables are as follows

1 Transaction
2Totals
3Issuers
4Acquirers
5Card Ranges

Reset

If reset is 1 then all fields in the column are reset.

CommStat Primitive

CommState(port, value, field)

port indicates the port to be. tested

value indicates a value to compare and set the status accordingly.

field (optional) indicates a field to store ComsState Value.

This function reads the state of the port store the value in field (if specified) and sets the current function result status to true if the value matches "value".

Date Primitive

Date(commdnd , date-field , time-field)

1Read system date into date field and time field

2Set system date from date field and time field

see also

Dates and Times

Dates & Times are special data types both are stored as special numbers.

Date Fields

A Date field is a two byte number, representing a date since 1Jan1940 to 1Jan2110. Subtracting two dates reveals the number of days between the dates, dividing by 7 reveals the day of the week (Monday=0, Tuesday=1 etc).

When moving to or from a text field a date is converted to a text format of DDMMYY. If a format is used this may be

converted to DD/MM/YY by using a currency symbol of / in the format. The text format of a date may be either 6 or B bytes long—showing the year as 2 or 4 digits.

Date Fields only contain valid dates. Since every date is stored as a day number, the storing the string 32/01/1980 will give is the actual date 01/01/1980. If data is entered directly into date fields, then dates are corrected in this manner automatically. If you wish to check the was entered correctly, then enter the data to a text field, then move the value to the date and check it is equal to the string.

e.g

Repeat

Print(GetDdte,O)

ActualDate:=TextDate

Until ActualDate==TextDate

The above example will continue to ask for a date until a valid date is entered.

Time Fields

Time fields may be either two or three bytes representing either the time of day to 2 seconds (two bytes) or 1/100 of a second (three bytes) accuracy.

Moving a time to a 2 byte integer gives the number of two second periods elapsed this day. Moving to a 1 byte integer extracts the 1/100s second fraction (up to 199)

Moving to a value or larger integer extracts the total number of tics (1/100s sec) which have occurred prior to the time.

Moving a time to of from a text field results in either HHMMSSFF when moving to a 8 or more byte field and HHMMSS when moving to/from a six byte field.

This text value may be formatted with a format to give HH:MM:SS.FF

Moving values between data and time fields and other numeric types results occurs without conversion. Moving to and from text values results in conversion. See specific entries for conversion details

Dial Primitive

Dial(phone numberphone number)

The numbers specified must be fields. Immediately following each number field in the data dictionary must be a timeout field then a retry field and then a mode field. The upstream prot is implied.

Do Primitive

Do (Function)

also known as

DoFunc(Function)

This primitive is used to activate another script function as a subroutine call

Event Primitive

Event(Function,system event)

Sets the specified function to be activated whenever the event occurs

Exit Primitive

Exit(Now?, Value)

This primitive is used to set the return value of the current function, and optionally, exit immediately.

The 'Value' is stored in the Calculation Result, which will be regarded by any calling function as a result.

If 'Now" is true (is 1) exit will be immediate, otherwise the exit value will be established.

Func Number Primitive

Function#ffieldlnumber. bad number function)

Execute the function with the assigned function#. Typically this primitive will be used by a user function set to be activated by a Hot Key on the target device labelled "Fn" or "Function" or similar. Such a user function would prompt for a number and then call this primitive (Function#) with that number as a parameter. See Function Activation.

The "Bad—Number—Function" is a function in the script to be executed if the no function matching the first parameter exists.

This is used to implement number functions—for example clearing memory might be function 1055. The user presses the "Function" hot key, then enters 1055 to execute the function.

To achieve this

1a function containing this primitive should created and set up under function activation to have the appropriate key code.

2A function containing the appropriate action for the numbered function should be created and set up under function activation to have the appropriate function number

The function number also returns the logical result of the request to call the numbered function. i.e false if no function exists, otherwise true.

If Else End Primatives

The next primitive is executed. If true then all primitives between the if and the else are executed. If false all primitives between the Else and End are executed. If nothing is required between for false then Else may be omitted.

If ! (if with <not?> parameter)

Else

Optional in an If see above.

End

Marks the end of an If or While. See While End

KeyBd Primitive

KeyBd(mode)—(O=off/I=on)

Upon entering idle state, the action of the keyboard is determined by the last KeyBd command. The keyboard (except cancel) will be ignored if off was specified.

MAC Primitive

mac(key,mode,message,field)

All targets must support the storing and use of 4 64 bit keys.

mode 1

Calculates a mac of the 'message' and stores the value in the 'field' specified. Uses the 'key' specified. If the 'message' is 8 bytes in length only (or less) then a single DES encryption only results.

mode 2

Stores the specified key into secure memory from 'field'

Mod

Mod (Value,Divisor)

The value "value" id divided by the divisor, and the remainder is the result.

e.g.—the following example would set the Data Field "Remainder" to 4. (25 divided by 7 has a remainder of 4.

Mod(25,7)

=> Remainder

Pin Primitive

pin (field)

Retrieves pin block from pinpad. Not supported on all CardScript devices

Print(Display/Report, Part)

Form

The Display/Report may be selected from the list box—or alternatively by selecting Field->Value[X] taken from a field in the data base.

Part

Values—0=all, 1 pre print/header, 2=body, 3=post print/footer

see Forms—end of Header/PrePrintt & Start of Footer/Post Print

Action

The selected section (or all) of the display is displayed or, in the case of a report, printed.

With displays, any input fields will be accepted, however the Show Primitive is recommended for input operations

ProcDown Primitive

Procdown(protocol, port)

The specified protocol is started on the port specified. This function is normally used for downstream protocols such as an ECR. This function is intended for more advanced users and the protocol must specify its own success and fail functions.

Protocol Primitive

Prot (protocol, Function 1, Function 2)

The specified protocol is started on the bank coms (upstream) port. The current function execution continues. KeyBd(Off) is set (it may be overridden). If the protocol returns a value of zero, Function 1 will execute, if any other value is return Function 2 will execute. (A KeyBd(On) will automatically happen before either function.

Range Primitive

Range(field,Min,Max)

Returns true if the value specified is >=min and <=max value

Repeat/Until Primitives

Repeat

Repeat sets the execution point for a following until

Until(Case) Cases are 0=False, 1=True

Executes the next primitive and if the result agrees with Case then the Repeats everything after the repeat primitive.

Report (Form, Function)

Action

First prints any pre print or header from "Form". Then for each transaction in the batch calls "Function" and prints the details section of "Form". After cycling throughout the batch, then prints any post print data from "Form".

Form

The Display/Report may be selected from the list box—or alternatively by selecting Field->Value[X] taken from a field in the data base.

Function

A function to be executed before each detail section is printed. For any transaction in the batch for which the function returns FALSE will be skipped.

Restore Primitive

Restore(layout,field,secondary field)

This primitive is used to retrieve information from the Batch Area.

Layout

This optional ("none" is permitted) parameter specifies which transaction layouts are considered for retrieval.

WARNING! All records searched using a field other than RECNUM are actually retrieved, changing the contents of the data fields in their layout. Using a value of "none" may have side effects!

Field

The primary search field. Searching will advance through the Batch Area until a match is found.

Secondary Field an optional (Use RECNUM for "none") secondary search field.

Rom Function Primitive

Rom(valuelfield, Message)

Generally the parameter passed is passed directly though to the bios. The following values are assigned for portability. The message parameter is ignored unless otherwise stated.

1Go to ROM mode.

2Erase memory and return to Rom

3Start TIVIS download (from ROM mode).

4Store Rom Params.

Uses Message and returns Success status.

See ROM SETTINGS.

51-oad Rom Params.

Uses Message and returns Success status.

See ROM SETTINGS.

6Activate Rom Edit.

Returns Success status. See ROM SETTINGS

see Also

ROM SETTINGS

What are ROM SETTINGS

Sub Heading

Normally target devices store programs in RAM memory, and are capable of loading these programs over the telephone Network. In order to achieve remote loading the device must store telephone number and other details required. The device must have a method of loading and/or editing these details.

Methods vary from device to device with information normally being obtained from the keyboard, a smart or magnetic card or some combination. Obviously the information must be able to be set prior to the application loading.

Communication Between Rom & CardScript

Two possible reasons for CardScript to interact with the ROM Settings arise.

1 The parameters may need to changed in a device already loaded with the CardScript application.

2A CardScript application may need access to the ROM Setting values.

Edit Rom Settings—Rom Function 6.

The desired method of allowing change to the settings is to use this function primitive call. (see Rom Function Primitive.) This primitive may not be supported by all Drivers and (check with the driver provider) but provides the only device independent method of implementing the function. An advantage of this function is the operator sees the same interface as when configuring the terminal prior to loading CardScript.

Load Rom Settings—Rom Function 4.

This function is used to obtain the ROM settings in a Script.

Store Rom Settings—Rom Function 5.

A Script Program may load the Rom Settings with Function 4, allow editing of values and use this function to store the settings. It is recommended to use function 6 (edit) in place of this prAc6edure where available as this mechanism allow changing of device specific settings.

The Rom Communications Buffer.

To provide a much device independence as possible using functions 4 and 5 CardScript defines a standard Communications Buffer Layout with a private area at the end. All Fields are ASCII.

The first three fields are assumed to be used for all communications. 2 bytes connection mode. Lan Leased Line etc 4 bytes station/Lan Address

8 bytes telephone prefix—eg "9,"

Field 1 16 Bytes Terminal ID. The ID as seen by the software management system and not necessarily other systems.

8 bytes terminal type

24 bytes phone number

24 bytes connection string

Save Primitve

Save(transaction layout)

Saves the current transaction to the batch using the layout specified. A new Transaction Index is generated according to the method specified by the last TxnIdx: Primitive, with the new index stored in field(0,0) RECNUM. For details on RECNUM see Reserved Data Dictionary Settings.

The number of transactions (of the selected layout) which can be stored is returned. If zero is returned, then the save

could not work! If 1 (one) is returned then no more may be saved. If two is returned then only one more may be saved, etc.

Store Primitive

Store(offset,messageLayout)

The store Primitve stores the last received message, starting at byte <offset>, using the specified message layout. The function result status is set by the operation. (Set to FALSE if the store did not match).

Tots Primitive

Tots(valuer Field)

Selects the relevant totals column.

This primitive has been replaced by the ColSelect primitive. Old programs are automatically upgraded, since parameter 2 or LineTble, when zero, will select the totals table

TxnIdx Primitive. Set Transaction Index.

TxnIdx(Field1,Field2,mode)

Field1 is optional. If include the first two digits of the Index are set from this field.

Field2 specifies the main field on which the Index is based. By default this is the ROC field.

the mode specifies how CardScript increments the Txn Idx.

0=add 1

1=Amex Style

2=None. Incremented by script.

This function would normally only ever be used in a start up function. The calculated value is always stored in the ROC field.

User Function Primitive

The user function primitive is used to call any of a range of functions. The functions call by user function are NOT standard.

Primitive—Extensions

It is possible to extend the primitives available to CardScript. The extensions take to form of a block of 'C' code loaded with the Script. 'C' code, of course, has the restriction of being non portable.

The existence of these extensions is to allow extensions to a set of primitives to be tested without changing the core driver. Any extensions initially tested by this means must be added to the set of primitives in a new release, otherwise the code calling them will never be portable.

Wait Primitive

wait(minutes, 100 msecs)

The current function pauses for then number of minutes+10ths of seconds specified. A delay of up to 1000 minutes (over sixteen hours is possible) and as small as 1/10 of a second.

While/End Primitives

While(Case) Cases are 0=False, 1=True

The next primitive is executed. If the result matches Case then all primitives between the While and the End are executed, then we come back again to the While. If the result does not match Case then execution continues with the primitive following the End.

End

Marks the end of an If or While. See also—.—If End

For specific categories of primitives see

Communications Primitives

Data Entry Primitives

Displaying and Printing

For information on configuring CardScript for target device function primitives (advanced users only) see

Configure Function Primitives

For advanced users only!!

Usage—Name Changes

Changing the name of a primitive or a primitive parameter will cause all scripts using the name change to be automatically updated. Both this type of change and any changes to the "infix" status of a function will have no affect on the driver and scripts will function without further change.

Usage—Adding, Deleting, Changing Parameter Types

Parameter Settings

Each function parameter has the following possible categories

1A Field from the data Dictionary

2Numeric Value—Which may be displayed as an index to a file

3A String

Any parameter may legally accept any combination of categories

Layouts

CardScript allows you to graphically enter your layout specifications. For details on Receipts, 'Reports, Displays, Messages, Protocols, and Transactions see Layouts Menu

Layouts are the main engine of any application. Although all layouts must bee brought into operation by functions, layouts also in turn launch functions and other layouts.

Form layouts, message layouts, and transaction layouts are similar in operation. All three are an arrangement of fields and strings called a field panel. For details see

Field Panels

All field Panels (Displays/receipts, messages and transactions) have a Panel Control box in common. The selection in the Panel Control box selects the action to take place when the left mouse button is clicked over the panel.

Additional controls are present of some panels, however, this box always contains

An add field—control with field edit box and palette selector

Clicking on the p'a'nel when [add] is selected adds a new field as displayed in the edit box

Before—clicking on the edit box to set the field to be added, select the appropriate type of item in the "from palette" drop down list box

Clicking on the edit box brings up either the Select Field Dialog or the Enter String Dialog, in accordance with the palette selector

A dealt field control

Select [delete] and then click on the appropriate field

A select field Control

Select [delete] and then click on the appropriate field

Field Editing

To edit any field, double click on the field

Forms (Displays and Reports)

see also Field Panel, Print Primitive, Show Primitive and Report Primitive

The Screen is Divided into four sections

The Form (Display/Report) Panel

The panel is a Field Panel where the location of the of each field corresponds to the place actual data will appear on the display/printer

The Dashed Boundary

Depending on the Display/Report type, a dashed line will appear showing the limits of the display or printer. This boundary is drawn in accordance with the settings in the Tables/System menu and can be changed at any time.

Form Name

The display report name is used for reference to this screen and should contain a meaningful name.

Panel Control

In addition to the panel controls discussed in Field Panel, two additional controls are present.

<<End of Pre Print

Pre-Print fields appear with a grey background

Select this item and click on the field after the end of the header section of the report.

Used in reports, the header section is printed once at the beginning of the report. The sections following the header will be printed once for each transaction in the batch. see Report Function for further details

Used in receipts (see Print Function for further details) used to divide the receipt not sections

Start of post print

Post-Print fields appear with a grey background, and can only be distinguished from Pre-Print fields if there are fields in between. (As would normally be the case.)

Select this item and click on the first field of the post print section of the report.

Used in reports, the Post-Print section is printed once at the end of the report. see Report Function for further details

Display/Report Type

The types are

Display—layout will always appear on the display, and in scribe will have a border reflecting display width and number of lines

Secure Display—reserved for future use

Printout—layout will always appear on the printer, and in scribe will have a border reflecting printer width

Soft Keys

The Soft Keys Button allows selection of a soft key set. see

Messages

Output messages

A messages buffer is built from fields in the data dictionary, and from strings in much the same way a printout is build. However, in messages, all data may be represented in forms other than ASCII. (see "the message engine". Formats may be used to specify data within the selected representation.

Input Messages

Messages are also used to specify how data is transferred from a received buffer and stored in data fields.

see also

The Message Engine (Processor)

The message engine is used both to transfer information both from data fields not a message buffer, and from a message buffer to data fields

see also

Message Data Mapping

Every field in a message buffer is converted from the type in the data field, to the representation of in the message buffer. For "Forms" all data in the buffer is Ascii, but in other messages the Oata may be any of the following

Ascii

Ascii Representation

Strings and Text Fields

Strings & text fields are simply copied. If the source is shorter then the destination, spaces are used for padding

Integer

Integer to Asch

For one & two byte integers, the binary value is converted to its string equivalent and then formatted according to any format specified. Larger integer conversion may appear in a later release

Ascii to Integer

Again limited to 1 and two byte integers, the value of the text is calculated and stored in the integer.

Amount

As for integer.

Dates

Dates are converted to either DDMMYY or DDMMYYYY if the Ascii field is longer than 7. Formatting is applied on conversion to ASCII only.

Times

Times are converted to either HHMMSS or HHMMSSFF (where FF is the fractions of a second in hundredths) if the Ascii field is longer than 7. Formatting is applied on conversion to ASCII only.

Hex

Hex Representation

Amounts

To Hex: The data is converted to a BCD string and then expanded

Integers

The binary value is converted directly to Hex. eg a one byte value set to 35 decimal (23 hex) would be converted to two bytes—characters' (Ox32) and '3' (Ox33) representing the hex value 23.

Binary

Binary Representation

Integer

Binary representation of integers is High Byte . . . Low Byte. As a binary value. No Actual conversion takes place

Text

Binary representation of Text is to assume the text is a hexadecimal string and convert this to binary. To get an exact copy of the string use Text Representation.

BCD

BCD Representation.

The Data is converted to the BCD data type.

Formats

Formats are used for specifying exactly how data will be represented

Justification

Any time the data length is less than the field width, the justification will be used to decide where the data is placed.

Allowed Characters

Specifies the type of characters allowed during input.

Minimum Characters

Specifies the minimum characters allowed for entry to a field.

Maximum Characters

Specifies the maximum characters allowed for data entry, and the maximum displayed characters on output.

Input Window

A non-zero value in this field specifies input will occur within a window. e.g A 24 character text field may be input using a 1 0 character window because of limited display space. Note that only ten characters of the input would then be visible at any one time.

Input Validation

A function may be specified here to valid input using this format. The validation function may store the current input using the ->(res) primitive. See also the Console Primitive command 4.

Note that an input validation function MUST NOT do any displays, as the current display would be overwritten.

Suppress leading zeros

Check here to suppress leading zeros in numeric fields, or leading spaces in text fields

Decimal places

Select the appropriate number of places, and the character to use as the decimal indicator. Selecting the decimal (from '.' or ',') also determines the character for thousands separation. (The opposite character to the decimal is used for thousands.

Check the box to require keying of the decimal indicator during input. If this box is left unchecked, data 1. 00 (. as decimal) would be input as 1 00, cash register style.

Auto OK

If this box is checked, when the maximum characters are entered, input will be concluded.

Thousands

The thousands separator(as determined under decimal places) will be automatically inserted.

Password Mode

The first character of the currency symbol will be displayed in each position, in place of the actual character entered.

Currency Symbol

Specify if the currency symbol is to be displayed. One or two characters may be entered. If the value is two digits, the digits are legal hex digits then these will specify a the character. e.g. 41 would specify the characters', as would a single A character.

This field as has other uses.

The separator for date & time fields is the first character. For time fields the second character is used to separator the hundredths of seconds when displayed

Length Indicator

In addition to the data, the data length is to be included. The number of digits to use may be 1,2 or 3. The length may be before (pre) the data or trailing (post). As an alternative to a numeric length specification the currency symbol may be used to indicate the end of a variable length field

e.g.

length as 1

5abcde is a five character field length as 2

5abcde is the same field

currency symbol as ':' and use currency selected

abcde: is the same field again

Pad BCF with F

Check here for BEC fields of odd length to be filled with a trailing F nibble. If unchecked a leading zero nibble would be used.

Transactions

Two other layouts types are also available

Bitmaps

Protocols

Protocols describe message flow both from and to the target device. The top line specifies outgoing messages and the other lines display possible incoming results. A protocol consists of lines and sections.

Request Line

At the start of each section is a line 1 (optional for the first section) which describes the outgoing message. This is the request line.

Response Lines

Lines 2, 3 and above define actions to be taken when a response is received. These are the response lines. A response may be a data received or a time out. When a timeout occurs the first line with a timeout will be selected, any other line with a timeout will never be used. When data is received, all lines beginning with a message are tested to see if received message matches the requirements.

The first item on each response line must be either a message or a timeout.

Protocol Screen Editing

Adding Entries

Select the desired item to add, then click on the display at the desired location

Splitting Sections

A section may be split on line 1. Click below the line slightly to the left of the field to become part of the second section.

Adding to a Line

When adding fields click on the field that will be after the new field. To add to the end of a line click about 3 spaces beyond the end of the line. Always click on the desired line.

Inserting a line.

Click where the new line should start

Retrys/Skips

After entering a Retry, the retry field will be selected. Or you can select the retry later. Once a retry is selected the <<set retry>> and <<set skip>> commands can be used to set the points where execution should move in the event of either a retry or the retry count being exceeded. Note, retry can only move back and skip can only move forward. For those with colour displays, the retry arrow is green and the skip arrow is drawn as red. You can't put a retry/skip on line one.

Identifying Input Messages

The first field of each input line is used to select when the input is appropriate. The following possibilities are catered for

Control

The input line is selected when the a message begins with the specified character

Message

The incoming message is matched against the message specified.

Timeout

A timeout line will automatically be selected in response to an incoming timeout.

Function

If the function returns true the message is matched. The Store Function.

Functions Launced By Protocols

Within protocols it is possible to launch functions for various reasons, particularly to store complex messages and select options.

Such functions should NOT halt operation, either by WAIT(or for input or any other event. Should a function attempt to do so, subsequent functions launched from the protocol, including the "good" and "bad" functions, will execute before the function resumes.

Delay any inputs until the good or bad functions at protocol end. If you are an expert user and must do an input, make it the last command in the function and exercise caution.

Repeated Messages

A protocol may involve repeated messages. That is, after storing the data from an input message, another similar message will be received.

Fixed number of repeat messages

If the number of times a message is to be received is fixed then the following approach may be used

<Msg><Retry(nn)>

Where nn is then number of messages expected

Variable number of repeat messages

If the number of times the message will be repeated will vary, i.e a flag in the message indicates that a repeat message will follow, then the following technique is recommended.

use an input line with <Function><Retry(O)>

This will cause a loop whilst the function returns true. From the function, use the store primitive to save the data and return true if another message is expected

Layout Primitives

Layouts use the following elements as building blocks

Putting it all together

Build Menu

Build Target Group Files

Builds the font and conf files ready for program execution

Build Script as Fragment

Builds a reduced script for loading either onto a smart or through the communications network, for describing a particular operation which may be changed without loading a new program.

Build Secure Prompts

Builds the list of secure displays and associated strings for loading into a secure display.

Run Simulation of Build

Activates the terminal simulator program

see also

Files Produced By Build

Font Files

xxx is the number of the font. Currently always zero

fontlxxx.bll

Characters O . . . 127. Bytes are dots across. First byte is top row. If more than 8 dots across, then the next byte continues the dots

fonthxxx.bll

Characters 128.255, using the same format as theTfile

fontdxxx.bll

Characters O . . . 1 28. Bytes are up and down. First dot is top left (bit 0 of byte 1) then dots down the character.

fontuxxx.bll

Script Fragments

Script fragments are small scripts (usually <256 bytes) built separately to a main program. Theses scripts may then be loaded into the terminal (either from a smart card or as part of a message) in order to specify operation of changeable program feature

Examples of Fragments

A Fragment for User Authentication

A Smart Card could contain a program fragment specifying how the terminal should check the user of the card is the real owner. Then cards may be issued with varied scripts such as

Input & Check PIN

Print A Slip and request a signature Do nothing—no check

A fragment for communications protocol

A server could have a list of communications protocols of various networks. Then the terminal could dial the server and request the relevant fragment for a particular network (either because the terminal has no information on the network—or the existing protocol no longer functions), allowing the terminal to operate on a new or changed network with obtaining a complete new application.

Menus

File Menu

Configure Menu

The Configure menu is greyed on standard level Scribe. The functions available are for the use of advanced users only. Generally within an Organisation using CardScript either one master user will be placed in charge of setting configurations or configurations will be set by an external consultant.

The configuration options are

Configure Simulation

Host Comms

Select a comm port for the simulator to use for modem communications. If none are available select "none". Selecting "none" precludes testing comms facilities

Terminal Group

The Build process creates several build files one for each terminal group. The setting chosen here determines which build file will be used for simulation.

Cards

The simulator does not use a real card reader. Instead it supplies card data from this table. Enter card data as required for up to eight test cards.

To provide the simulator with information on this PC and to create test "Cards".

Configure Targets

A number of target machines may be described to the CardScript system. The information about the target machines is used in various places throughout the scribe system to present information in a manner appropriate to a currently selected "target" machine. see System Record information for setting a Current target.

Object types

The target machine is described by arranging an number of "objects" on this panel. Their is one of each of the display, printer, and reader objects, and as many button objects as are appropriate.

The display object is used to specify the display configuration in lines and columns. This object should be dragged to an appropriate location in the window.

The printer object is used to specify the print width columns. This object should be dragged to an appropriate location in the window. The number of lines setting bears no relation to the number of lines on a printer page, this field determines how many lines will be available for viewing during simulation.

The reader object is used to describe which area of the window will be used to display buttons for simulation of a card reader. This area bears no relationship than actual card reader. Draq this object to an otherwise unused area of the window.

Any number of button objects. The object correspond to the push buttons on the target device. Five different button styles are available. Configure these styles as required. When a style is changed, all buttons of that style will change in appearance. Keycodes returned should match those returned by the actual terminal BIOS. Use the KeVMap Primitive to force the map these codes to the codes required by the actual application.

For describing the various hardware platforms to the system

For designing Tables Screen Layouts and Contents used on the PC with Scribe

For designing the Data Dictionary used in the Target Device

For Specifying the functions available within the target device.

Reference

Index

Glossary

#

"–>" (goes to) primitive: <goes to primitive>

"KeyBd": <KeyBd Primitive>

"refer": <ColSelect Primitive>

"target": the PC, EFTPOS terminal, PiNpad or cash register which will be used to run the developed application.

Case: 14-1135 Document: 1928 Page: 198 Filed: 02/18/2014

->(res): <result goes to primitive>

B

Batch Area: Storage area of memory. Used for storing transactions and any other miscellaneous data. Also may be thought of as file storage.

bitnumbering: <Bit Numbering>

C

CardEntryMode: <Card Entry Mode>

ColSelect primitive:

Commands for Memory Cards: <Commands For Memory Cards>

Console Primitive: <Console Primitive>

D

Data Dictionary Field Attributes: <Data Dictionary Field Attributes>

Date & Time Fields: <Dates and Times>

E

eserved data dictionary fields:

F

Field Panel: <Field Panels>

Forms—end of Header/PrePrinft & Start of Footer/Post Print: <Forms>

Function Action: <Function Action>

Function Actions: <Function Actions.>

Function Primitive Categories: <Function Primitive Categories.>

Function Primitives: <Function Primitives>

H

HEX: . Digits O . . . 9 and A . . . F

Initial Data Field Attributes.: <Initial Data Field Attributes>

K

KeyMap Primitive: <KeyMap Primitve>

P

PFIELD: special field for use on Forms. In place of this field a supplied parameter field will be displayed

Print Primitive: <Print Primitive>

PSTRING: special field for use on Forms. In place of this field a supplied parameter string will be displayed.

R

RAD: Application Development (especially used with 'tool')The process of defining a program in a very short time by starting the program definition with the user interface.

Report Primitive: <Report Primitive>

Reserved Data Dictionary Settings:

Reserved Data Dictionary Settings: <Reserved Data Dictionary Settings>

ROM SETTINGS: <ROM SETTINGS>

S

Show Primitive: <Show Primitive>

Smart Card Type Code: <Smart Card Type Codes>

System Table Settings: <System Table Settings>

T

Txnldx Primitive: <Txnldx Primitive>

Windows: popular operating system for PCs. based on an event driven architecture.

Appendix B

Bios Objectives

The objective of the CardSoft BIOS is to make all devices used for running Point of Sale software compatible with CardScript programs.

Bios Usage

The CardSoft BIOS specification is designed to allow the creation of portable programs for Payment Terminals. Any given implementation of the BIOS will encompass its own "look and feel" which, in turn, is imparted to applications using the system. This is possible since the BIOS specifies what must be achieved by low level functions, rather that the manner of achievement, This means that not all implementations of the BIOS are equivalent and there is scope for vastly different performance and operational convenience whilst still maintaining BIOS compatibility.

As an example, the BIOS itself does not specify how such things as how cursors and editing functions are implemented, there is simply a call specifying display this field and allow it to be edited. Thus the field editing rules are determined by the individual BIOS implementation. One brand of equipment over type may be standard, on another insert may be the default.

This leaves individual implements with the ability for creativity and a framework which allows for the performance and convenience of their programmers to be a product advantage.

Also supplied in addition to the core BIOS are some implementation routines. These are supplied in source code as a starting point for actual implementation. However the code in these routines is not applicable to all hardware configurations and would expect over time to be modified in any given implementation.

The BIOS described in this manual represent the interface between CardSoft EFT applications (including the driver for CardScript) and an EFT terminal, however the specification is general purpose in nature and may in future be used to support other systems. This manual assumes CardScript is to be supported and is geared to assist in achieving this goal.

In addition to the functionality described here the EFT device must have its own "bootstrap" system. Where CardSoft applications are being added to existing products a software module which interfaces between routines described here and the existing driver software can easily be produced.

This BIOS specification remains the property of CardSoft.

Utilising Existing Operating Systems

When first adding CardScript to a device, some level of BIOS or operating system will normally be already in place.

In many cases it is desirable to add CardScript to devices originally developed years prior with well tested hardware device drivers. In these instances the BIOS will constitute an interface between the existing operating system and the CardScript driver. The BIOS may then be linked with the Driver and the combined application loaded as one conventional application.

The BIOS is designed to be able to be placed as a layer above any pre-existing operating system, and be loaded together with the Driver program an one application to devices installed in the field.

New Products

In the case of new products, created for use with CardScript, a purpose build BIOS will minimise memory requirements and speed time to market.

Steps To Implementation

The steps in implementing the CardSoft BIOS are as follows. Check the BIOS library supplied with this manual is correct for your microprocessor development tools. Other versions of this library for other development environments may be obtained from CardSoft.

Choose appropriate optional code. In order to simplify implementation of the BIOS sample code is supplied for some typical hardware configurations types providing

higher level functionality and simplifying installation. It is recommended to make use of this software initially, and replace code as desired once the system is operation on the target hardware. Use supplied outline "main.c" and compile & link.

Add routines to eliminate unresolved externals. Use empty routines supplied for routines to be supplied later. It is recommended to initially include real keyboard and display routines and then add others.

Concepts

Event Driven Structure

Applications constructed to run on the CardSoft BIOS must be event driven. This allows the BIOS or operating system to have control during idle periods. When an input event occurs, the application is called to process the input, and then returns to the BIOS/operating system. The application sets which routine will handle each message and what messages are enabled.

This event driven structure allows the BIOS to operate as an interface layer to event driven operating systems without problems. Where the underlying structure is not event driven This enables the BIOS functionality to be matched by either low level BIOS code or by a high level operating systems ensuring maximum portability of CardSoft application, and enabling sophisticated underlying structures to be utilised where present. The event driven structure of the system means that applications do not contain a "main" procedure. Applications have an init-application (routine which sets up a table of routines addresses to bo called in the case of external events occurring.

Callbacks

Callback Control—Input

The BIOS must maintain a callback address table with four entries for each input/output file. Associated with each entry in the table is an enable status. When the corresponding event occurs a callback should be made using the address from the table. Each callback contains an optional Code and Message (see below). The BIOS should not issue a second callback while another callback is in progress. This can lead to race conditions.

Output

For porting the CardSoft Bios to a new machine see

Low Level Interface

The low level interface represents the routines that must be custom written when porting CardScript to a new device.

These routines assume that the standard Console module, or equivalent are used. Use of these modules eliminates most of the work in implementing the CardSoft Bios, but postpones fine tuning the Bios to make use of Specific hardware in the most efficient manner.

Standard Modules

The following modules irriDliment the high level interface

console.c

callback.c

math.c

To implement the low level interface, a single module may be created interfacing the foiling routines to the actual hardware or existing drivers. the categories of routine are

Low Level Display

void dispbin(uchar ch)

Display character ch at current cursor position and advance cursor one place

void cleardisplay( )

void dispStr(uchar *str,int len)

continuous dispbin for length of str.

length of str is either len characters, or if len=-1, then str is null terminated.

uchar dispscroll(uchar direction)

Directions are

1 left

2right

3up

4down

Each call is a request to scroll one place in the specified direction. The result indicates the success of the request (1=OK, 0=can't do)

Low Level Printer

The only printer routine is low level. see

Printer

void prch(uchar ch)

This routine simply prints the character "ch" on the printer device.

Special codes are as follows

OxA (10) end of line

OxC (12) end of form. Feed lines as required for tear off of receipt

10 General Routines

uchar softKeyBase(uchar select)

By convention returns V for a parameter of 0, and 'a' for a parameter of 1. Change these to indicate actual key values for soft key sets

buz

void buz(int freq, int duration)

Communications

The following routines must have the code inserted to call the low level drivers correctly.

All routines work with a comfile number. Number 0 is the default and is used for the modem. Number I should be the auxiliary com port, if present. Number 2 is the second auxiliary (again if present) etc.

Sendcom_msg

This routine sends block of characters to the specified port. If low level drivers (such as those used with HDLC) require the block at one time then you will need to call those drivers directly from here. If the target device supports only character mode communications, then the "sendcom" routine may be called once for each character.

```
void sendcom-msg (int comfile, uchar *buf, int countOfChars)
{
}
```

Sendcom

A singe character is transferred to the specified port.

```
/* individual coms character send routine */
void sendcom (int comfile, int ch)
```

-continued

```
    {
    }
```

Dial

This routine is used to start the dial process. "num1" is to be dialled "cnt1 times, then if this fails, "num2" is to be dialled "cnt2" times. "Mode" indicates the communication mode to be used. These parameters are under direct control of the application programmer, but by convention mode1=async,2=HDLC

```
/* MODEM */
void dial (uchar *num1, uchar len1, uchar cntl,
uchar *num2, uchar len2, uchar cnt2, uchar mode)
{
}
```

Hangup

The equivalent of the ATH command on a hayes modem.

```
void hangup ( )
{
}
txstate ( )
```

uchar txstate( )

This return should return a status as follows

0=busy

1Ready

2Reserved for errors, no currently used

Real Time Clock

The real time clock is red & set with the biosDateTime( ) routine

biosDateTime( )

unsigned int biosDateTime(command, buffer)

Command (1=read Dateltime,2=set date & time from buffer Buffer DDMMYYYYHHMMSSFF

TPDU—The smart card interface

uchar driveTPDU(uchar 11, uchar 12,uchar *Command, uchar *sendBuf,uchar *receiveBuf)

This routine implements, both the Scribe TPDU and SmartCard primitives. To decide which is call is being made, the Command parameter must be tested.

Command==NULL, SmartCard Primitive

11, and 12 are the parameters. Refer Scribe.hlp for details

Command!=NULL—TPDU primitive

This a direct implementation of the scribe TPDU primitive, with L1 as the length of the sendbuffer, and L2 the length of the receive buffer.

If L1 is non zero, there is data to send to the card. If L2 is non ZERO, then data from the card is required.

Result

Return zero, unless the function is used as the SmartCard Primitive, and a result is required.

Timer

CardScript requires the target device to have a 100 millisecond timer. This timer should may a call to the script routine "time—tick( )"

It is recommended not to make a call direct from the hardware timer interrupt. This would result in actions launched by time tick( ) to execute with interrupts off, giving some very strange results.

Instead set a flag in the interrupt handler and have the event loop clear the flag and call time-tick( ) (if using an interrupt handler).

The script driver includes the routine "start-bomb( )" which may be called by the bios interface if required

The Font File

The font file consists of sets of entries as follows

1 Character code—1 byte

2Width—1 byte

31-leight—1 byte

4Bitmap

The bitmap is arranged as follows

For each row of height as many bytes as needed for the bits (1 for width <=8, 2 for width <=16 etc).

Left most bit in the MSB of the first byte.

The file is appended with a block of three zero bytes. (Code, width, Height=0) and no bitmap.

For fine tuning an operational Bios see

BIOS Specification (High Level Interface)

By Category Routines are:—

Console (Display & Keyboard)

Input

Sequential (Non event driven)

Since the non event driven-machine must be made to appear event driven, the bios interface must include the main line and call the application to handle any events

```
Sample main ( )
{
init - hardware( ); /* perform any hardware specific
initialisation */
init - applications ( ); /* call to routine in module DRIVE
*/
for (; ;)
{
    if(event) /* test for event */
    { clear - event ( ); /* clear event status */
    handle-event ( )*/ call CardScript event handle - see
list */
}
. . .
    }
}
```

Events and Handlers

The following list of events should be catered for

Events List

Console

If the high level console is used. Keyboard events are reduced to a single call-Process—

Process-key

void process-key(uchar key-code)

All that is required by the implementor is to may key codes from the actual machine to the those to be seen by the CardScript application.

Please note that the application programmer has the ability to remap the keys sing CardScript.

Special Key Codes

OxA "Enter" or "OK". (Completion of input)

Ox8 "Clr'or backspace

Ox1B Cancel

The only other console input is the magnetic card reader.

Please use

callback(Console,3,<unused>,buffer,<unused>);

(use 0 (zero) for unused parameters. or

process-card (<unused>, buffer)

Magnetic Card Road Buffer

The buffer may include any or all of the following sections. They must appear in order.

Section I (optional)

Identifier byte (001)

Track 1 Data—all ascii values

Track2 (optional)

Identifier byte (002)

Track 2 Data—all ascii values

Track 3 (optional)

Identifier byte (N03)

Track 3 Data—all ascii values

End of data marker

Identifier byte (0x0)

System

System Events

For each event, make a call to the routine

void systemEvent(uchar event)

Communications

Comms Events

Character comms

callback(Port,1,Character,NULL,O)

Message based comms

callback(port,2,0,BufferAddress.MessageLenght)

Dial or Tx Finished

When any operation which made the communications port busy has finished, it should tell the script driver by the following call

callback(port,4,0,NULL,0)

see also

Event Driven Input

Please consult CardSoft for further information

Structural

Memory Management

The MEMPTR type

External Memofy

Often target devices have 8 or 16 bit microprocessors which can address limited memory without the use of paging. To allow access to such memory, the concept of External Memory has been defined

It is not assumed that this external memory is directly addressable by the CPU, instead this memory is accessed only via the memory management functions.

The script, the data dictionary fields, any optional fonts and the file storage area are all stored in "external" memory. These areas of external memory are allocated numbers as used in the getbase(function,

A type MEMPTR is used to address this external memory. In the include file custom.h the type MEMPTR must be defined. If has less than 64 k of memory allocated amongst the external memory areas MEMPTR could be defined as unsigned integer. If the memory is larger than this than MEMPTR will normally be defined as "long".

Each block of external memory must APPEAR to be continuous. That is incrementing a MEMPTR with the C++ operator must always generate a pointer to the next byte of the area.

The memory management routines must map these virtual memory addresses in to real memory addresses

getbase(base)

The function getbase returns the virtual memory address of each of the following blocks of memory

1 The smart card execution buffer

2The Script area—(includes the initialised data dictionary tables

3The Uninitialised data dictionary table area

4The file/batch area

getDataByte

Returns the byte at the virtual address

uchar getDataInt(offset)

Returns the two byte value at the virtual address specified. The format of the two byte value is always Low/High regardless of the byte ordering of the microprocessor.

getScriptData

void getScripfflata(uchar *bufferMEMPTR offset,int size)

Transfers data from the buffer to the virtual address "offset"

setScriptData

void setscripfflata(uchar *buffer,OFFSETTYPE offset,int size)

Either set external memory to NULL bytes or to a copy of a buffer buffer is the memory buffer in the standard memory area OR if NULL then the operation is like a memset

add notes on offset type

size is the number of bytes to store

For customising the CardScript command set

Adding Function Primitives

All function primitives have up to four parameters. Each Parameter is of either one or two bytes length.

Numeric value parameters of values O . . . 1 27 are one byte in length. Numeric values of greater length and in the general format, with a maximum value of 4999.

All other parameters are in the general format, sixteen bits High order first bit 15 set—numeric value all other 15 bits contain the value high nibble=Ox5 next three nibbles give string number.

all other values high byte=table, low byte=field.

Reference

Index

Glossary

#

"start bomb( )": <start-bomb>

"time-tick( )": in the script-driver for processing 1/10 second time ticks

What is claimed is:

1. A communication device which is arranged to process messages for communications, comprising a virtual machine means which includes

a virtual function processor and function processor instructions for controlling operation of the device, and message induction means including a set of descriptions of message data;

a virtual message processor, which is arranged to be called by the function processor and which is arranged to carry out the message handling tasks of assembling the messages, disassembling messages and comparing the messages under the direction of the message instruction means that is arranged to provide directions for operation of the virtual message processor, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task,

wherein the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems.

2. A device in accordance with claim 1, further comprising a virtual protocol processor arranged to organize communications to and from the device, and

protocol processor instruction means arranged to provide directions for operation of the protocol processor means.

3. A device in accordance with claim 2, wherein the device includes a microprocessor which runs in accordance with native software code and the protocol processor is implemented as a native software code of the microprocessor.

4. A device in accordance with claim 2, wherein the device includes a microprocessor which runs in accordance with native software code and wherein the protocol instruction means are implemented in software defined by the protocol processor means, and do not require translation to the native code of the microprocessor.

5. A device in accordance with claim 1, wherein the device includes a microprocessor which runs in accordance with native software code, and the message processor is implemented as the native software code of the microprocessor.

6. A device in accordance with claim 5, wherein the function processor is implemented as native code of the microprocessor.

7. A device in accordance with claim 1, wherein the message processor instruction means is implemented in software defined by the message processor, wherein the device includes a microprocessor, and wherein the message instruction means do not require translation to the native software code of the microprocessor.

8. A device in accordance with claim 1, wherein the device includes a microprocessor which runs in accordance with native software code, and wherein the function processor instruction means are implemented in software defined by the function processor means and do not require translation to the native code of the microprocessor.

9. A device in accordance with claim 1, including a hardware abstraction layer comprising a series of routines which provide an application program interface to exercise an operating system, BIOS or hardware drivers of the device.

10. A device in accordance with claim 1, wherein the device is a specialized network access device arranged for communication over a network.

11. A device in accordance with claim 10, the device being a remote payment terminal and the messages being messages relating to remote payment transactions.

12. A method of programming a device for processing communications, comprising the steps of loading a processing means of the device with a virtual machine which includes a virtual function processor and function processor instructions for controlling operation of the device, and a virtual message processor which is arranged to be called by the functions processor and which is arranged to carry out the task of assembling, disassembling and comparing messages, under the direction of the message instruction means that is arranged to provide directions for operation of the virtual message processor, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task, wherein the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems.

13. A method in accordance with claim 12, comprising the further step of loading the processor means of the device with a virtual protocol processor arranged to organize communications to and from the device, and protocol processor instructions arranged to provide directions for operation of the protocol processor.

14. A computer memory storing instructions for controlling a computing device to implement a virtual machine means which includes a virtual function processor and function processor instructions for controlling operation of the device, and a virtual message processor which is arranged to be called by the function processor and which is arranged to carry out the task of assembling, disassembling and comparing messages, under the direction of the message instruction means that is arranged to provide directions for operation of the virtual message processor, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task, wherein the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems.

15. A computer readable memory in accordance with claim 14, further storing instructions for implementing message processor instruction means arranged to provide directions for operation of the message processor.

16. A computer readable memory in accordance with claim 14, further storing instructions for implementing a virtual protocol processor arranged to organize communications to and from the computing device.

17. A computer readable memory in accordance with claim 16, further storing instructions for implementing protocol processor instructions arranged to provide directions for operation of the protocol processor means.

* * * * *

Case: 14-1135 Document: 28 Page: 203 Filed: 02/18/2014

Patent No. 7,302,683, Dated November 27, 2007

US007302683B2

(12) **United States Patent**
Ogilvy

(10) **Patent No.: US 7,302,683 B2**
(45) **Date of Patent: *Nov. 27, 2007**

(54) **METHOD AND APPARATUS FOR CONTROLLING COMMUNICATIONS**

(75) Inventor: **Ian Charles Ogilvy**, Manly (AU)

(73) Assignee: **CardSoft International Pty Limited**, Manly NSW (AU)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/207,771**

(22) Filed: **Aug. 22, 2005**

(65) **Prior Publication Data**

US 2006/0015646 A1 Jan. 19, 2006

Related U.S. Application Data

(63) Continuation of application No. 09/381,143, filed on Oct. 22, 1999, now Pat. No. 6,934,945.

(30) **Foreign Application Priority Data**

Mar. 14, 1997 (AU) PO9896
Mar. 16, 1998 (WO) PCT/AU98/00173

(51) **Int. Cl.**
G06F 17/00 (2006.01)

(52) **U.S. Cl.** 718/1; 703/23; 703/26; 703/27; 709/200; 709/201; 709/208; 709/219; 719/310; 719/313; 719/315; 719/316

(58) **Field of Classification Search** 718/1; 719/310, 313, 315, 316; 703/23, 26, 27; 709/200, 201, 208, 219

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,336,870 A	8/1994	Hughes et al.	
5,479,643 A	12/1995	Bhaskar	
5,553,291 A	9/1996	Tanaka et al.	
5,590,281 A	12/1996	Stevens	
5,708,838 A	1/1998	Robinson	
5,745,886 A	4/1998	Rosen	
5,923,884 A *	7/1999	Peyret et al.	717/167
5,926,631 A	7/1999	McGarvey	

(Continued)

FOREIGN PATENT DOCUMENTS

AU 2200088 9/1988

(Continued)

OTHER PUBLICATIONS

"Smarter Smartcards," Hofland & Janowski, Feb. 1998, Byte Magazine 401S, pp. 7-10, whole document.

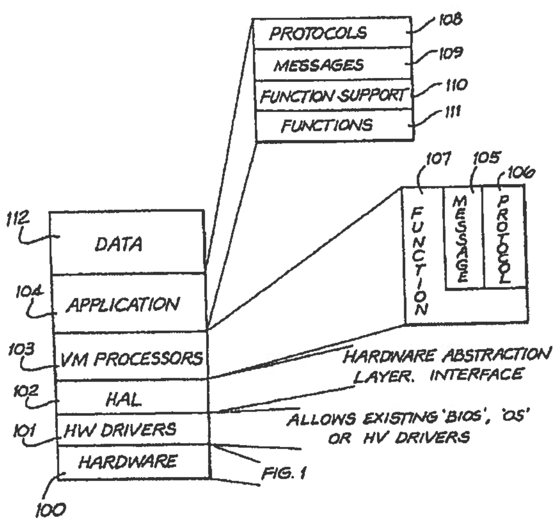
(Continued)

Primary Examiner—Phuoc Nguyen
(74) *Attorney, Agent, or Firm*—Duane Morris LLP

(57) **ABSTRACT**

Disclosed is a device arranged to process messages for communications, comprising a virtual machine means including a message processor means which is arranged to process messages communicated to and/or to be communicated from the device, and message processor instruction means, arranged to provide directions for operation of the message processor means. Also disclosed is a method for operating a device arranged to process messages for communications and a method of programming a device arranged to process messages for communications.

5 Claims, 12 Drawing Sheets



U.S. PATENT DOCUMENTS

5,931,917 A	8/1999	Nguyen et al.	
5,935,249 A	8/1999	Stern et al.	
6,021,275 A *	2/2000	Horwat	717/159
6,199,160 B1	3/2001	Echensperger et al.	
6,308,317 B1	10/2001	Wilkinson et al.	
6,934,945 B1 *	8/2005	Ogilvy	718/1

FOREIGN PATENT DOCUMENTS

EP	0456249	11/1991
EP	0634718	1/1995
WO	9212480	7/1992
WO	9305599	3/1993
WO	9410628	5/1994
WO	9727537	7/1997

OTHER PUBLICATIONS

"Java and the Shift to Net-Centric Computing," Hamilton, IEEE Computer, Aug. 1996, pp. 31-39, whole document especially "Java Virtual Machine," sidebar p. 32.

* cited by examiner

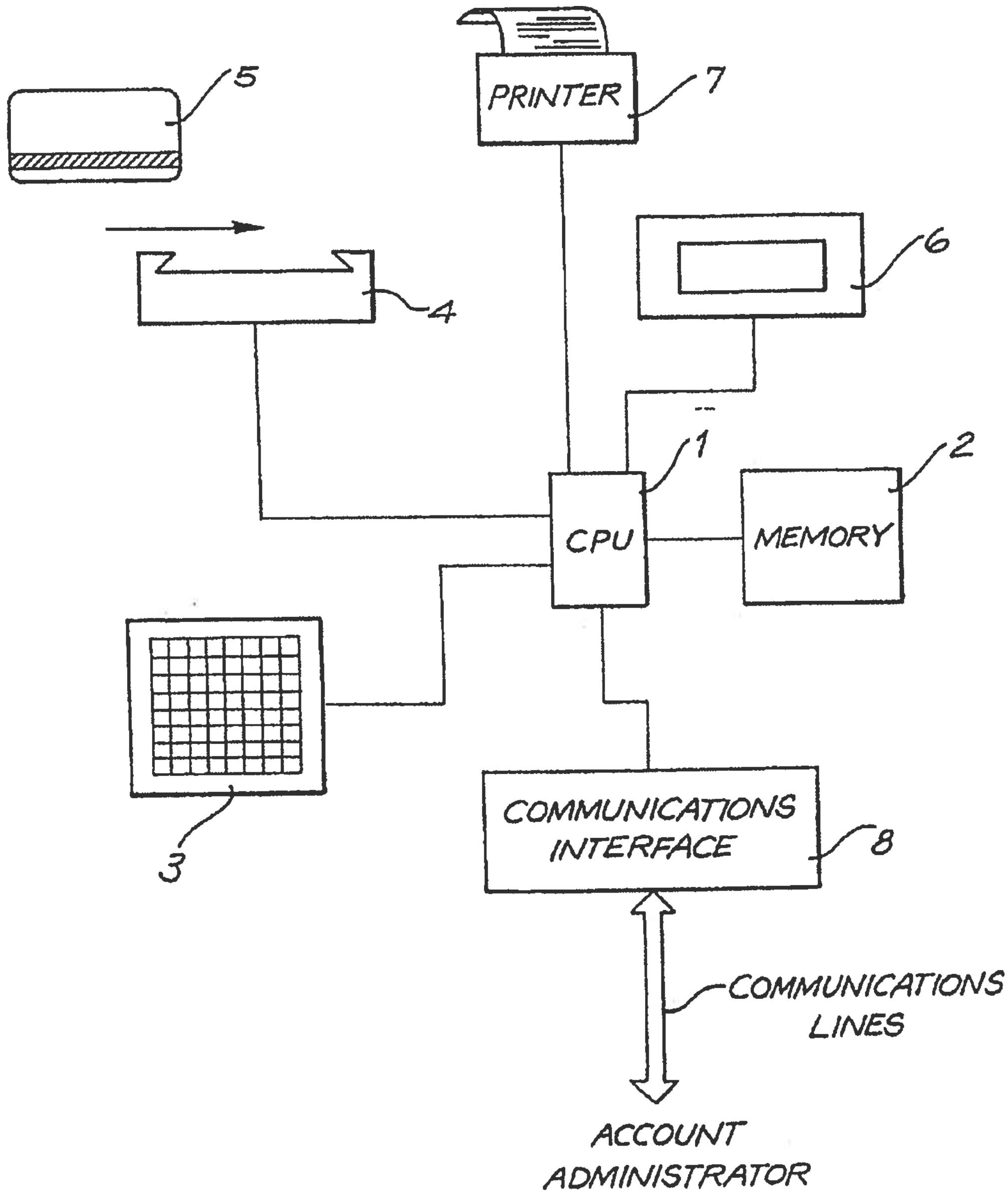

5
PRINTER
7
6
4
1
2
CPU
MEMORY
3
COMMUNICATIONS INTERFACE
8
COMMUNICATIONS LINES
ACCOUNT ADMINISTRATOR

FIG. 1

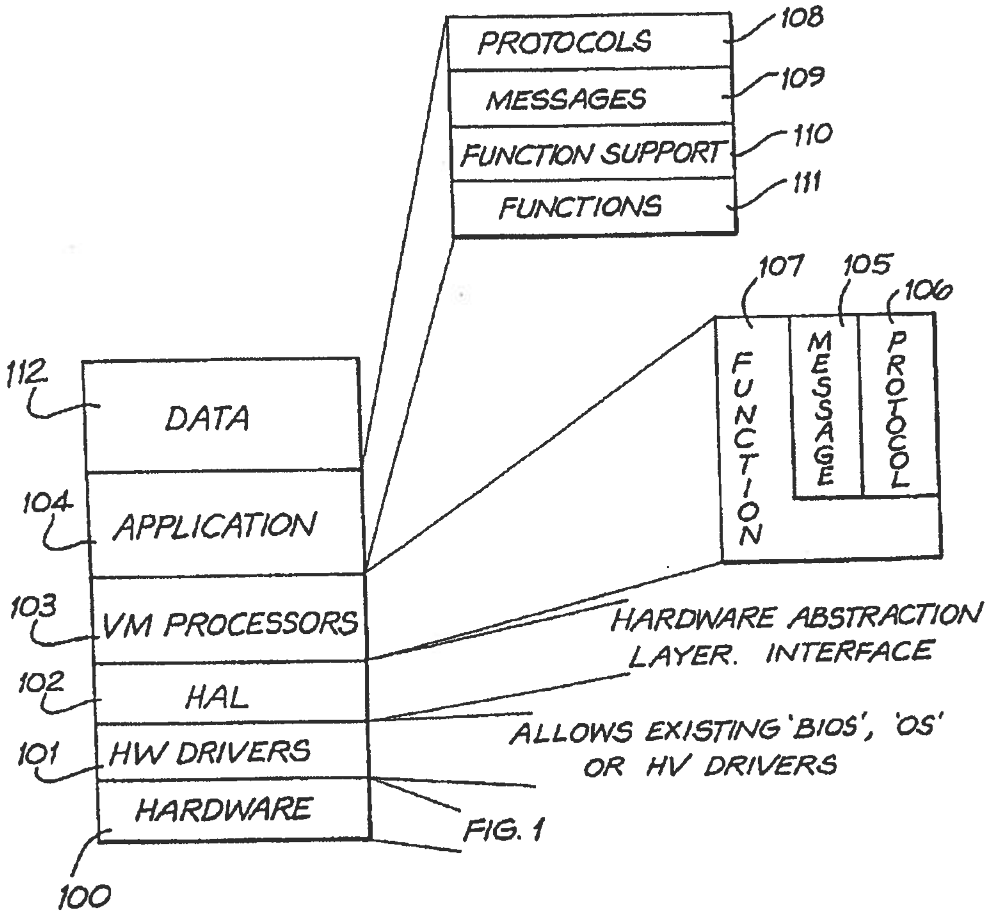

PROTOCOLS
MESSAGES
FUNCTION SUPPORT
FUNCTIONS
108
109
110
111
107
105
106
FUNCTION
MESSAGE
PROTOCOL
112
DATA
104
APPLICATION
103
VM PROCESSORS
102
HAL
101
HW DRIVERS
HARDWARE
100
HARDWARE ABSTRACTION LAYER. INTERFACE
ALLOWS EXISTING 'BIOS', 'OS' OR HV DRIVERS
FIG. 1

FIG. 2

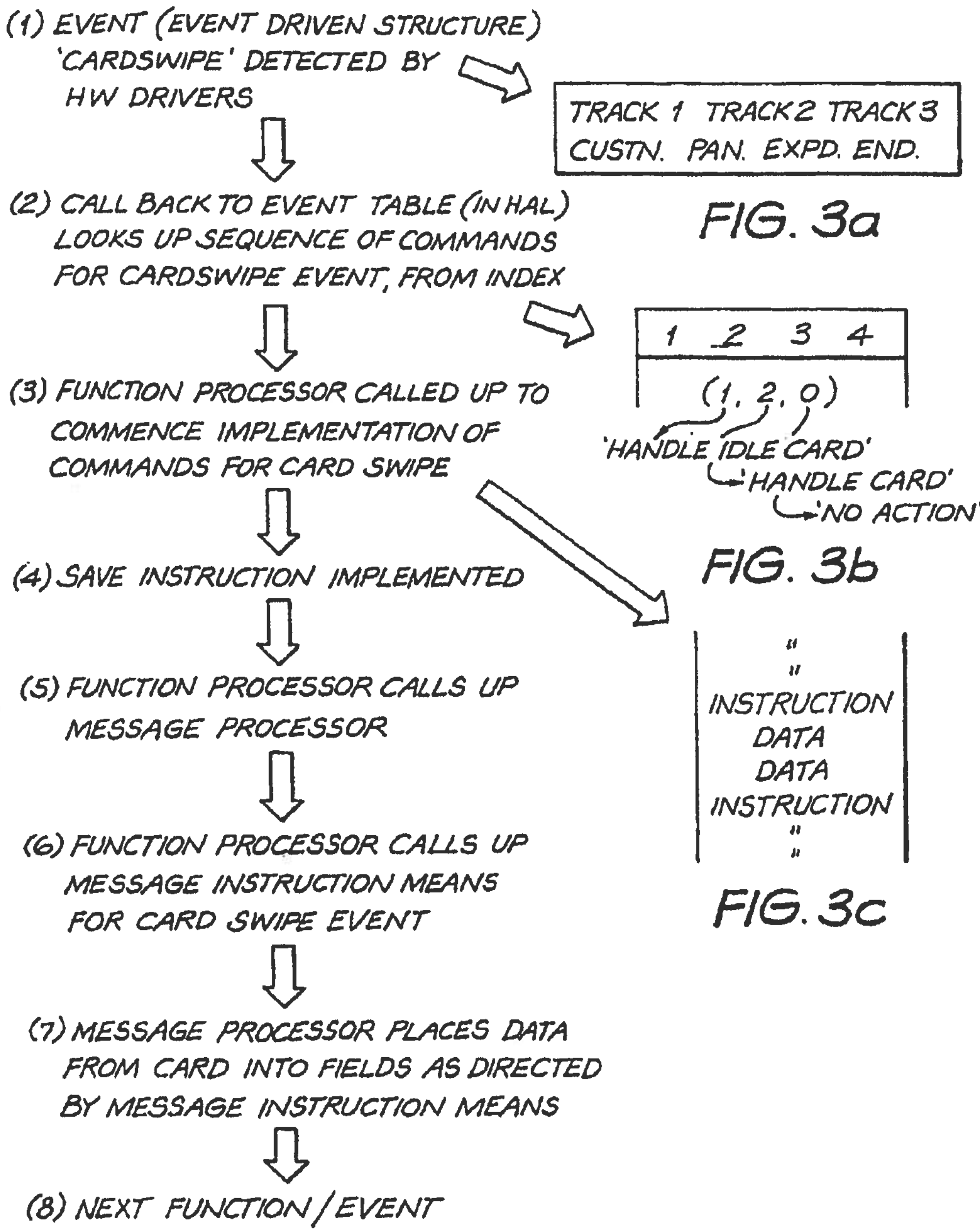


FIG. 3a

FIG. 3b

FIG. 3c

FIG. 3

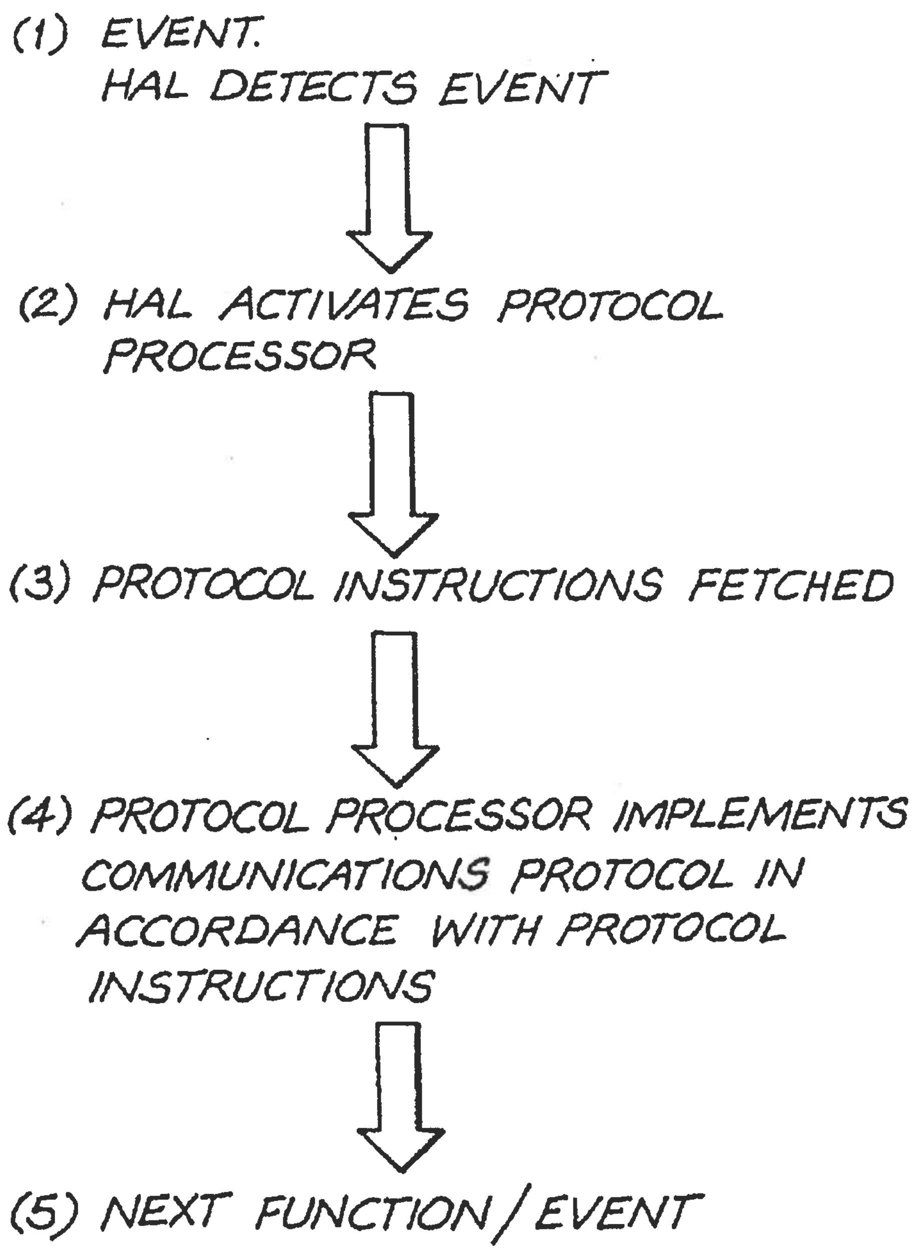

(1) EVENT.
HAL DETECTS EVENT
(2) HAL ACTIVATES PROTOCOL PROCESSOR
(3) PROTOCOL INSTRUCTIONS FETCHED
(4) PROTOCOL PROCESSOR IMPLEMENTS COMMUNICATIONS PROTOCOL IN ACCORDANCE WITH PROTOCOL INSTRUCTIONS
(5) NEXT FUNCTION/EVENT

FIG. 4

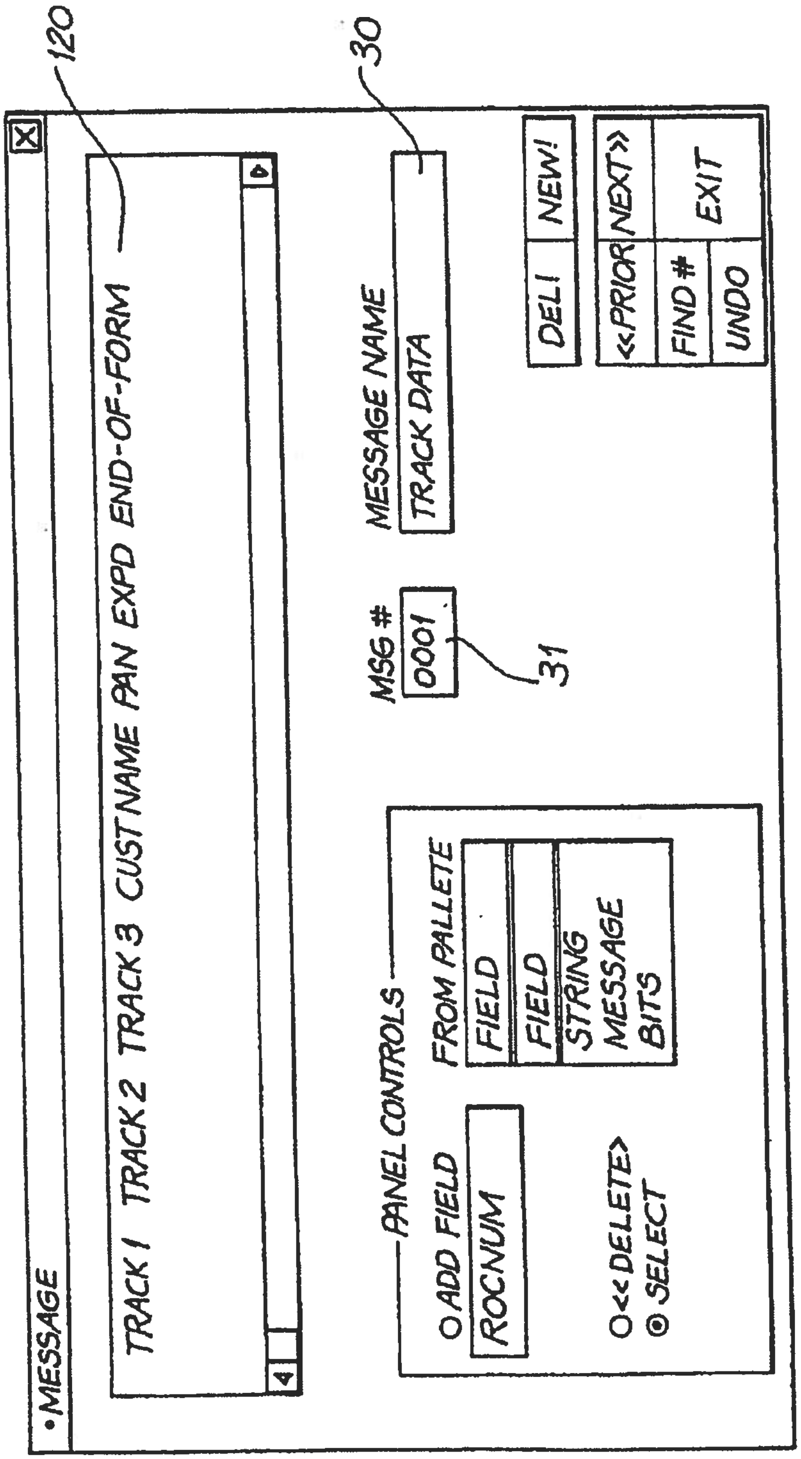

120
30
•MESSAGE
TRACK 1 TRACK 2 TRACK 3 CUST NAME PAN EXPD END-OF-FORM
MESSAGE NAME
TRACK DATA
MSG #
0001
31
DEL! NEW!
<<PRIOR NEXT>>
FIND # EXIT
UNDO
PANEL CONTROLS
ADD FIELD
ROCNUM
FROM PALLETE
FIELD
FIELD
STRING
MESSAGE
BITS
<<DELETE>
SELECT

FIG. 5

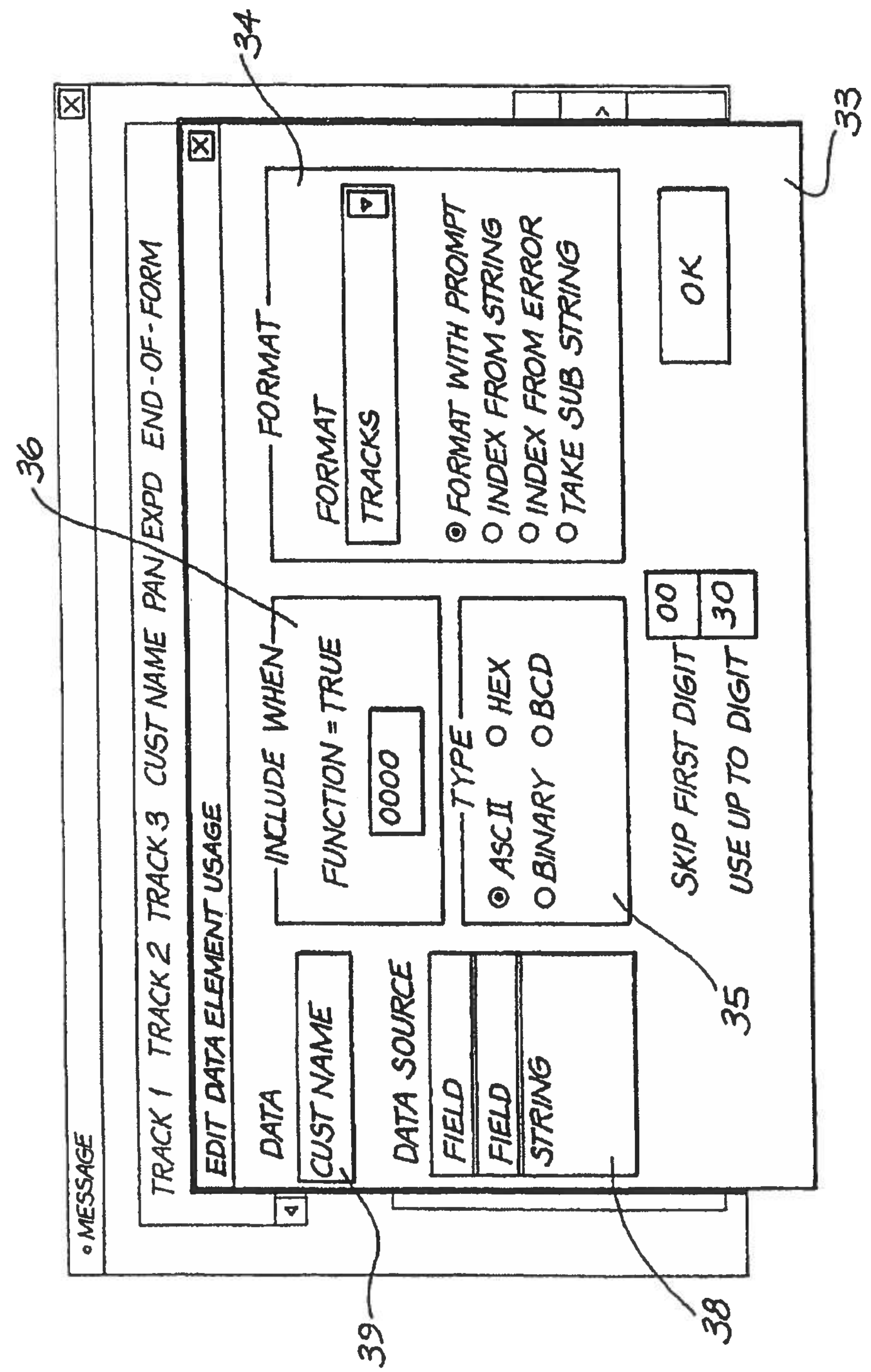

MESSAGE
TRACK 1 TRACK 2 TRACK 3 CUST NAME PAN EXPD END-OF-FORM
EDIT DATA ELEMENT USAGE
DATA
CUST NAME
DATA SOURCE
FIELD
FIELD
STRING
INCLUDE WHEN
FUNCTION = TRUE
0000
TYPE
ASCII
HEX
BINARY
BCD
FORMAT
FORMAT
TRACKS
FORMAT WITH PROMPT
INDEX FROM STRING
INDEX FROM ERROR
TAKE SUB STRING
SKIP FIRST DIGIT
00
USE UP TO DIGIT
30
OK
33
34
35
36
38
39

FIG. 6

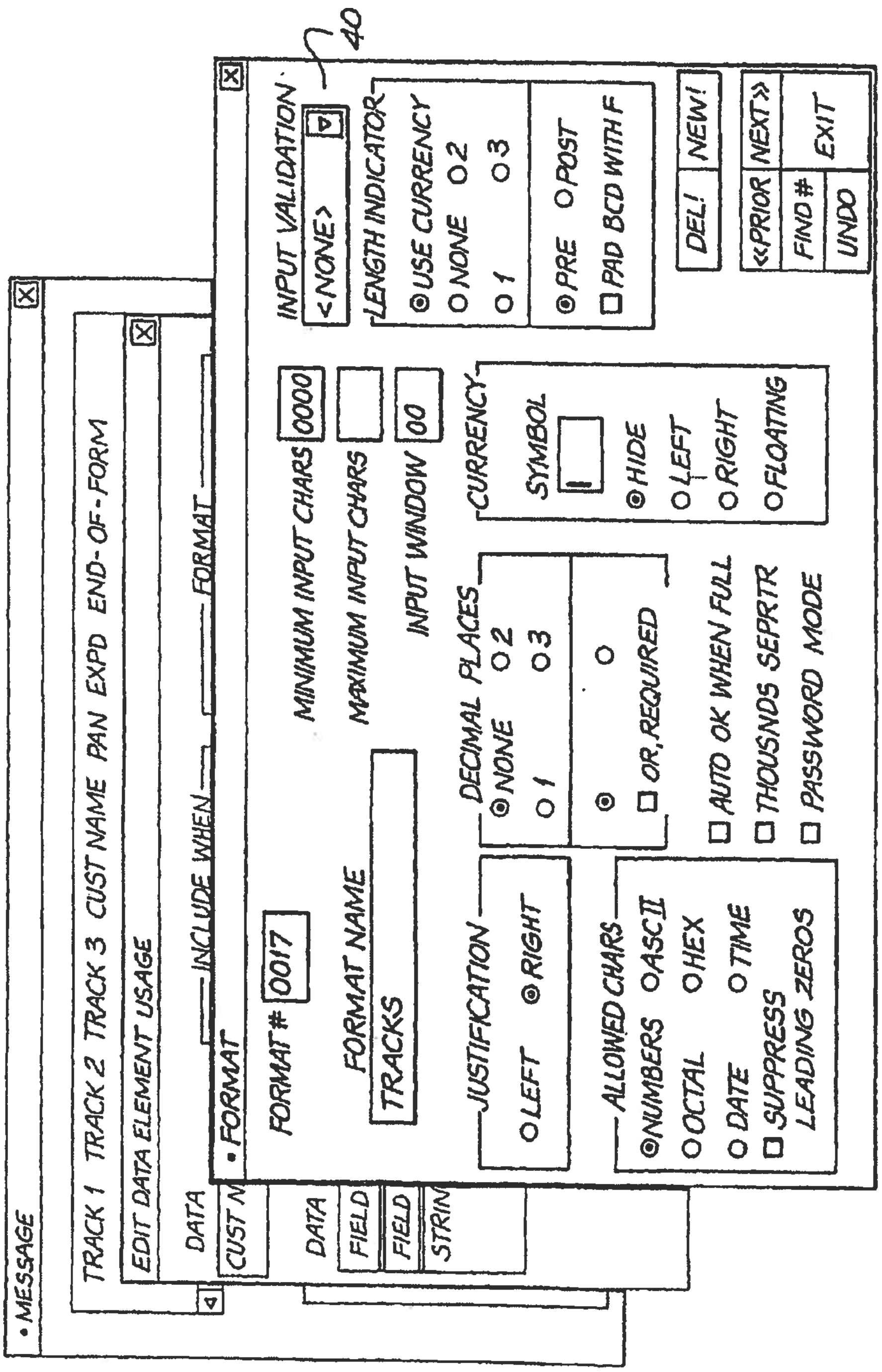

• MESSAGE
TRACK 1 TRACK 2 TRACK 3 CUST NAME PAN EXPD END-OF-FORM
EDIT DATA ELEMENT USAGE
DATA
INCLUDE WHEN
FORMAT
CUST N
DATA
FIELD
FIELD
STRIN
• FORMAT
FORMAT # 0017
FORMAT NAME
TRACKS
JUSTIFICATION
LEFT
RIGHT
ALLOWED CHARS
NUMBERS
ASCII
OCTAL
HEX
DATE
TIME
SUPPRESS LEADING ZEROS
MINIMUM INPUT CHARS 0000
MAXIMUM INPUT CHARS
INPUT WINDOW 00
DECIMAL PLACES
NONE
2
1
3
OR, REQUIRED
AUTO OK WHEN FULL
THOUSNDS SEPRTR
PASSWORD MODE
CURRENCY
SYMBOL
HIDE
LEFT
RIGHT
FLOATING
INPUT VALIDATION
<NONE>
40
LENGTH INDICATOR
USE CURRENCY
NONE
2
1
3
PRE
POST
PAD BCD WITH F
DEL!
NEW!
«PRIOR
NEXT»
FIND #
EXIT
UNDO

FIG. 7

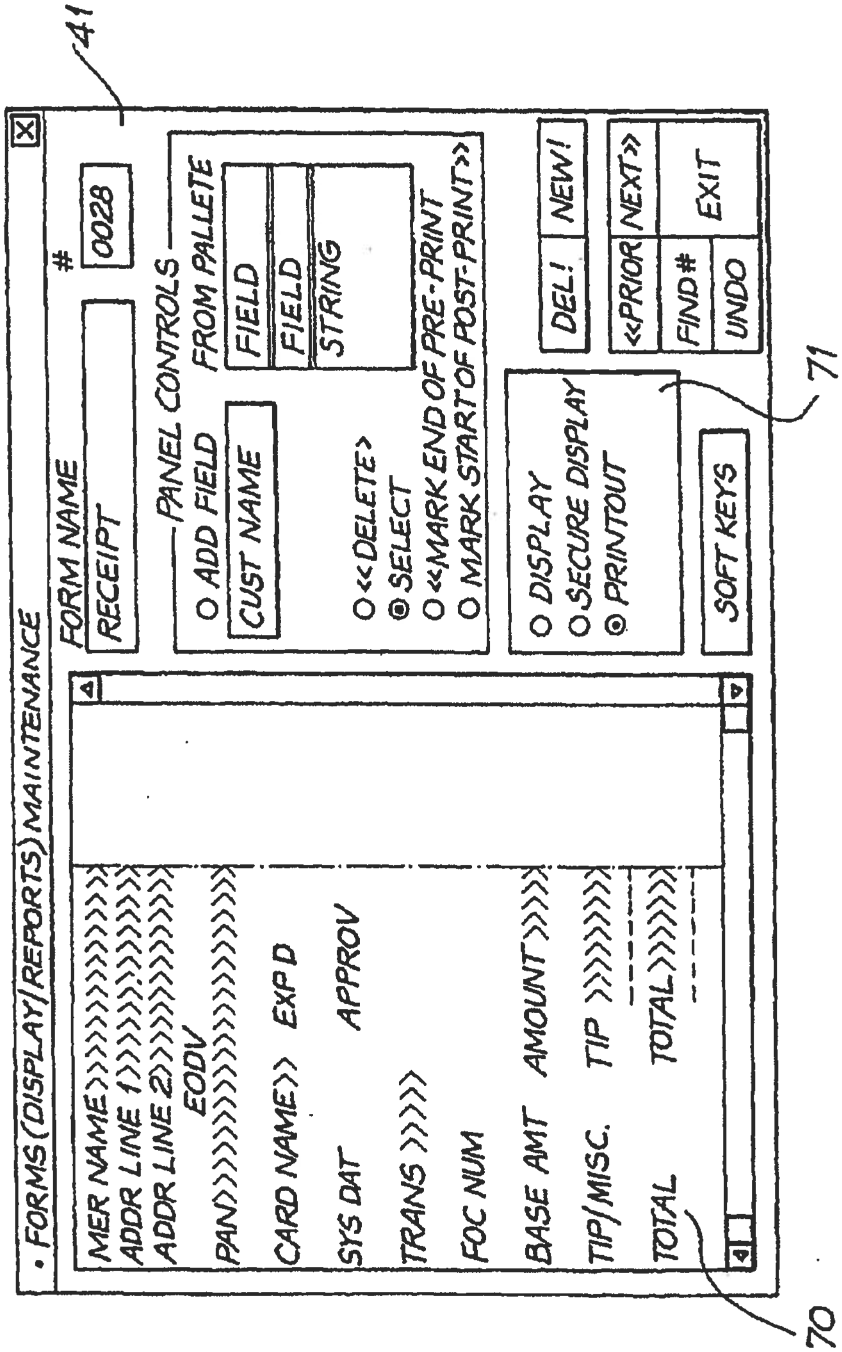

41
FORMS (DISPLAY/REPORTS) MAINTENANCE
FORM NAME
RECEIPT
#
0028
PANEL CONTROLS
ADD FIELD
FROM PALLETE
CUST NAME
FIELD
FIELD
STRING
<<DELETE>
SELECT
<<MARK END OF PRE-PRINT
MARK START OF POST-PRINT>>
DEL!
NEW!
<<PRIOR
NEXT>>
FIND #
UNDO
EXIT
DISPLAY
SECURE DISPLAY
PRINTOUT
SOFT KEYS
71
MER NAME>>>>>>>>>>>>>>
ADDR LINE 1>>>>>>>>>>>>
ADDR LINE 2>>>>>>>>>>>>
EODV
PAN>>>>>>>>>>>>>>>>
CARD NAME>> EXP D
SYS DAT APPROV
TRANS >>>>>
FOC NUM
BASE AMT AMOUNT>>>>
TIP/MISC. TIP >>>>>>>
TOTAL TOTAL>>>>>
70

FIG. 8

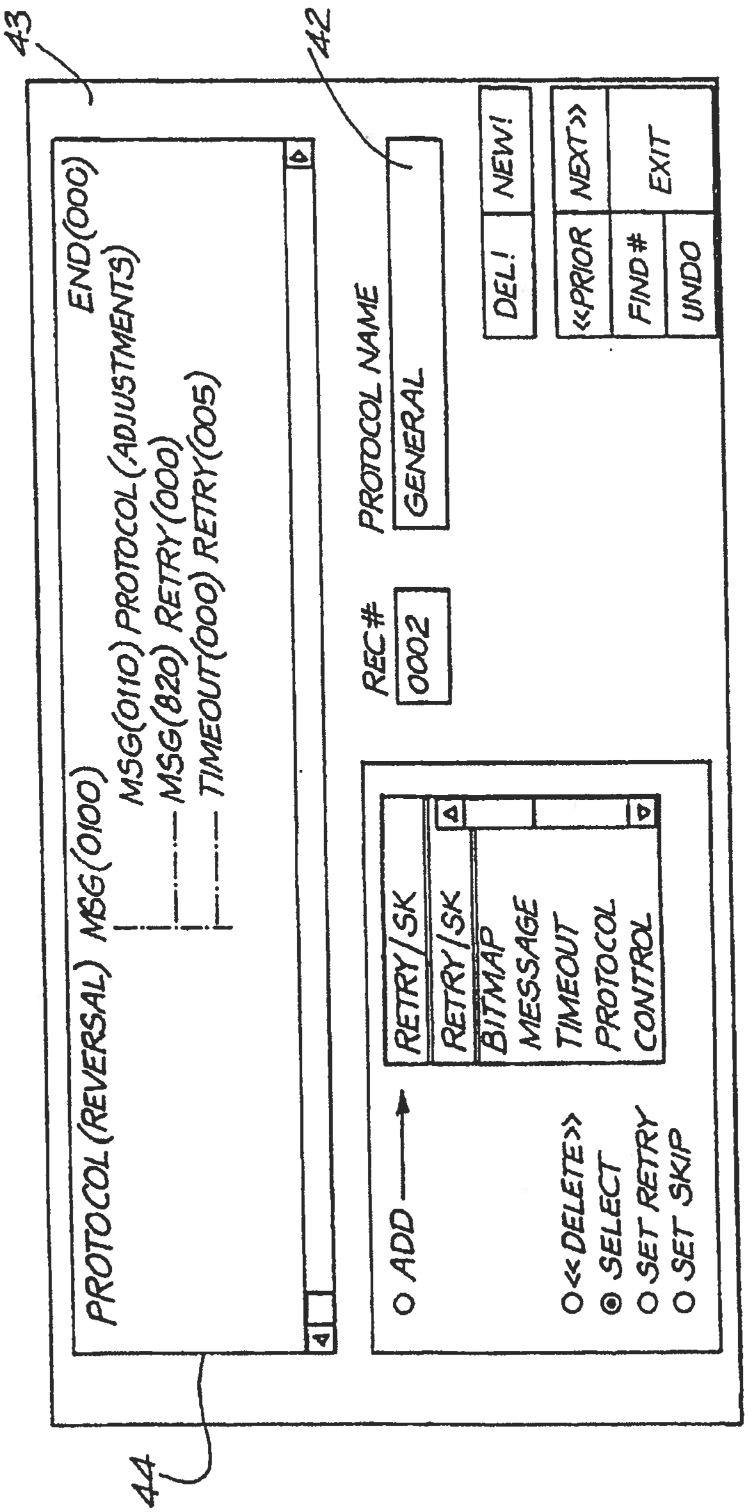

43
42
44
PROTOCOL (REVERSAL) MSG(0100)
MSG(0110) PROTOCOL (ADJUSTMENTS)
MSG(820) RETRY(000)
TIMEOUT(000) RETRY(005)
END(000)
PROTOCOL NAME
GENERAL
REC #
0002
DEL!
NEW!
«PRIOR
NEXT»
FIND #
UNDO
EXIT
ADD
RETRY/SK
RETRY/SK
BITMAP
MESSAGE
TIMEOUT
PROTOCOL
CONTROL
«DELETE»
SELECT
SET RETRY
SET SKIP

FIG. 9

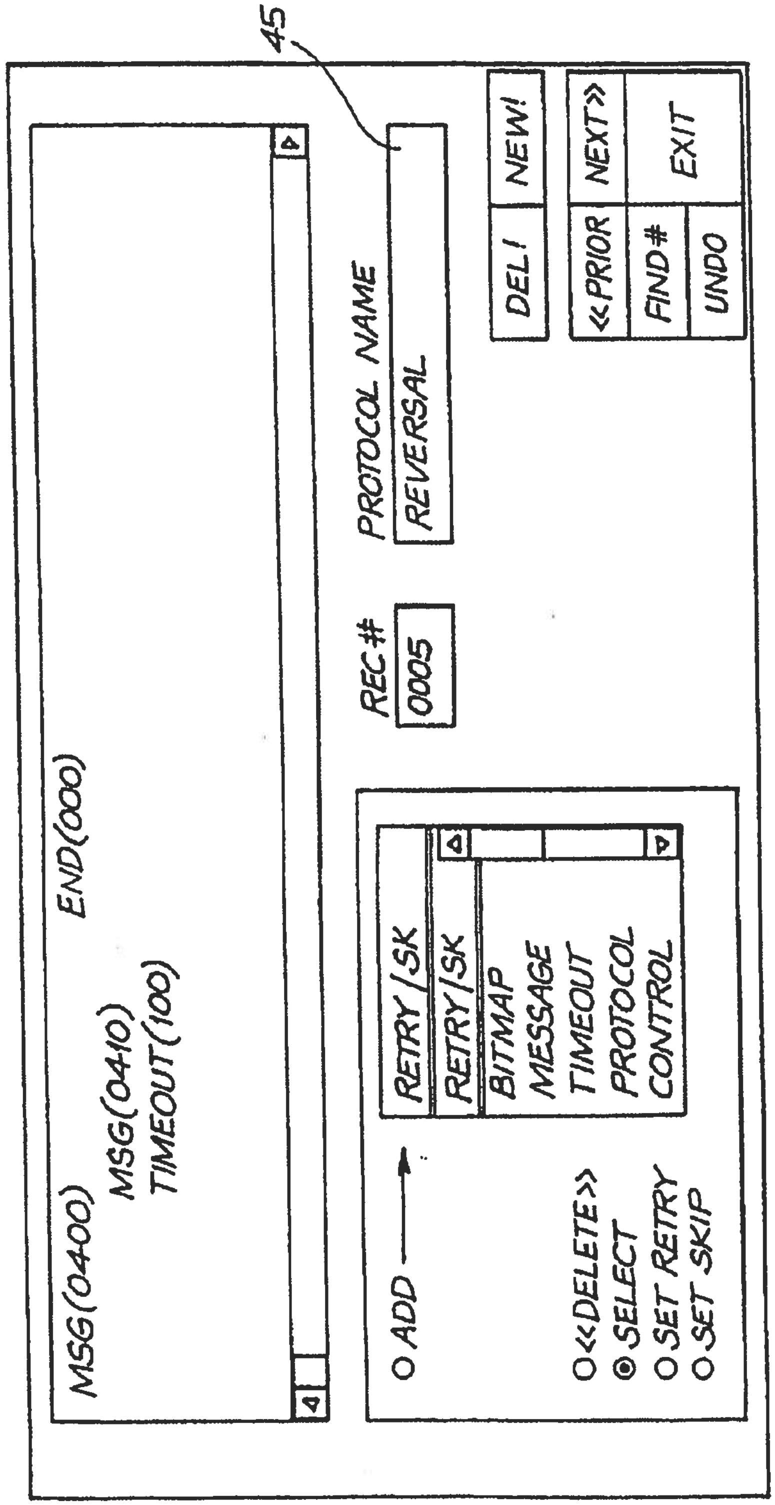

45
MSG(0400)
MSG(0410)
TIMEOUT(100)
END(000)
REC #
0005
PROTOCOL NAME
REVERSAL
DEL!
NEW!
«PRIOR
NEXT»
FIND #
EXIT
UNDO
ADD
RETRY/SK
RETRY/SK
BITMAP
MESSAGE
TIMEOUT
PROTOCOL
CONTROL
«DELETE»
SELECT
SET RETRY
SET SKIP

FIG. 10

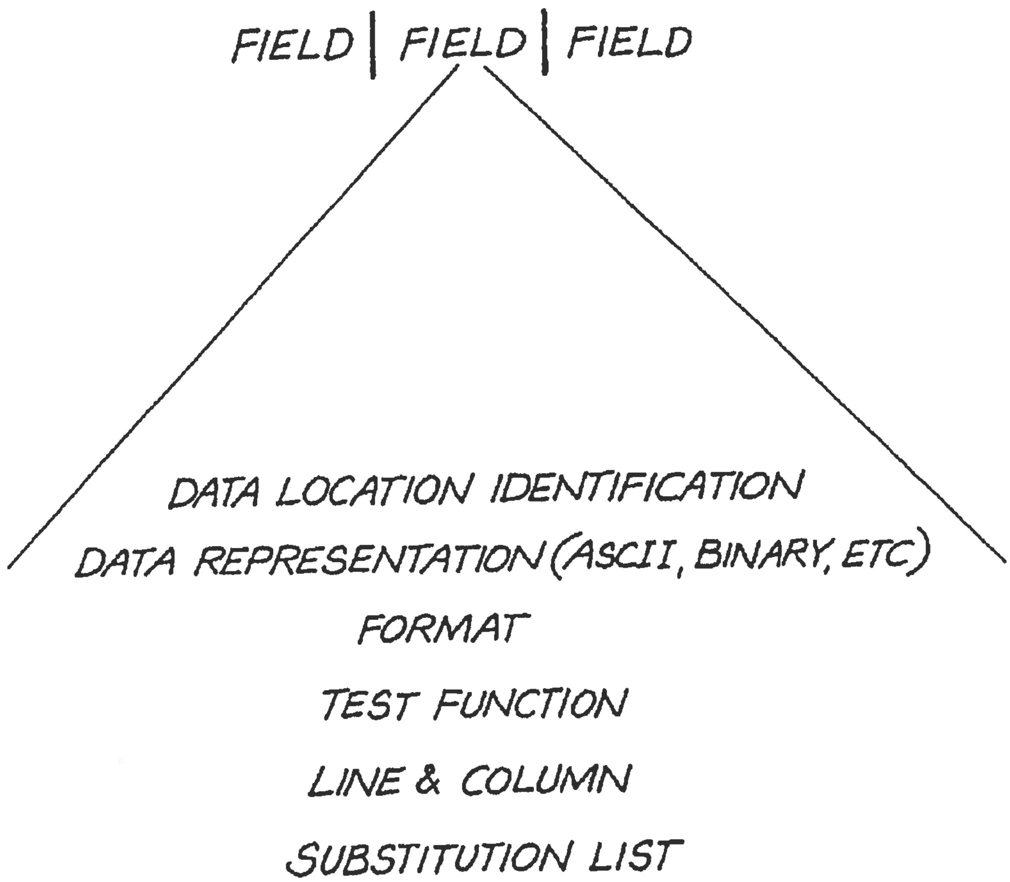

FIELD | FIELD | FIELD
DATA LOCATION IDENTIFICATION
DATA REPRESENTATION (ASCII, BINARY, ETC)
FORMAT
TEST FUNCTION
LINE & COLUMN
SUBSTITUTION LIST

FIG. 11

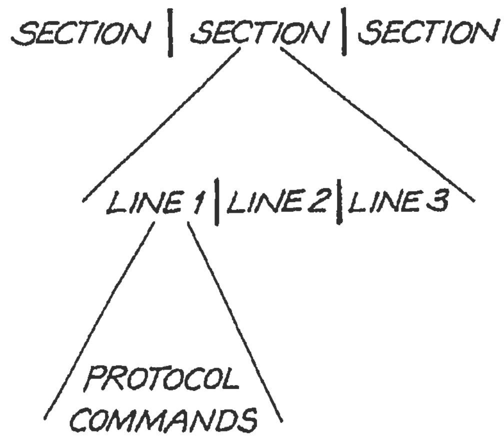


FIG. 12a

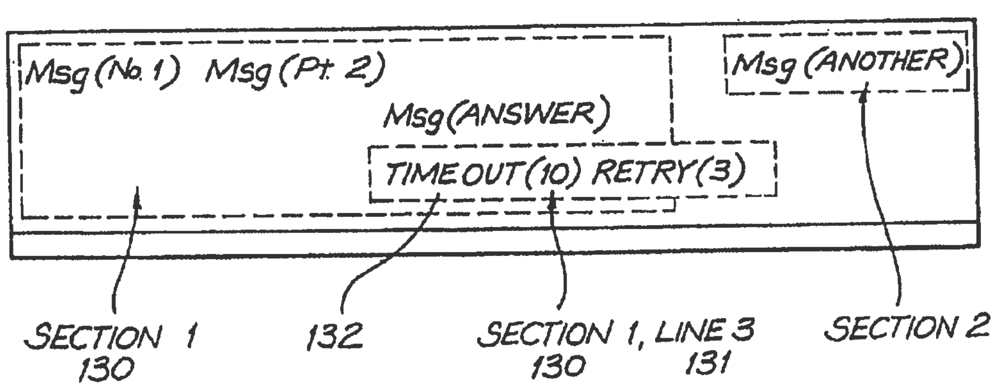


FIG. 12b

METHOD AND APPARATUS FOR CONTROLLING COMMUNICATIONS

This application claims the benefit of the PCT Application No. PCT/AU98/00173, filed on Mar. 16, 1998, which is hereby incorporated by reference in its entirety. The present invention is also related to U.S. application Ser. No. 09/381, 143, filed on Oct. 22, 1999, now U.S. Pat. No. 6,934,945 which is hereby by incorporated by reference in its entirety.

FIELD OF INVENTION

From a first, general aspect, the present invention relates to a method and apparatus for preparing and processing information to be sent or received via a network. A network in this instance may be implemented as data carried either over communications lines and/or stored on smart cards (or other data carriers) and physically transported.

From a second, more specific aspect, the present invention relates to a method and apparatus for controlling remote payment transactions, particularly, but not exclusively, for controlling remote payment transactions where a persons account is credited and/or debited from a remote location in exchange for goods/services cash or credit, or where account information is accessed remotely to enable approval of a transaction.

BACKGROUND OF INVENTION

Devices for carrying out remote payment transactions are well-known. These "payment terminals" include EFTPOS, credit card payment terminals, etc.

The most common function of payment terminals is to remotely access a persons account information and either carry out a transaction, such as crediting or debiting the account, or, particularly in the case of credit card payment terminals, to check the users account to ensure that there are sufficient funds to cover a transaction. Note that although credit card terminals do not necessarily remotely credit or debit the users account (the credit/debit transaction usually being carried out by a separate paper bill trail) and merely provide the information that the users account is sufficient to cover the transaction, such payment terminals still fall within the ambit of the present invention and the term "transaction" as used herein includes the operation of remotely checking the users account to "ok" a transaction.

A payment terminal may, for example, provide for the following basic operations:

(1) Input of information which is required to enable access to a customers account. The information is most often read from a magnetic stripe on a credit card or bank card or the like, or a smart card. In addition to reading details from a card a personal identification number (PIN) or the like code may also be required.

(2) Obtain access to the customers account. This is usually done by remote communication with a processing device holding the person's account data, usually on bank premises and remote from the payment terminal. Usually, information on the customers account input to the payment terminal will need to be transmitted for verification and to enable access to the account. Also a money amount will usually need to be input to the payment terminal and transmitted over the communications line. At least some and perhaps all of the transmitted data may be encrypted for security purposes and the payment terminal is therefore, in such a case, required to have means (3) providing encryption.

(4) The payment terminal may need to be able to receive communications over the remote line from the processor accessing the customers account, ie. to provide an "answer" to the payment device regarding the user transaction. The answer may include information that an account debit/credit has taken place (eg. EFTPOS) or merely an approval that the customer has enough money in his account to enable a transaction (some credit card payment terminals). Again, this transmitted information may be encrypted and, if so, will require translation (5) in the payment terminal.

(6) To provide an indication that the transaction request is approved or that a transaction has occurred, by display or printer, for example. Displays may also be required to prompt an operator or customer to input information, e.g., input your PIN "Input Amount".

There are many different brands of payment terminal, utilising many different software and hardware arrangements. This gives rise to a number of problems.

Any account acquirer (eg. bank) will generally have their own operating requirements as to how remote payment transactions will be handled. The account acquirer may purchase a series of payment terminals which have been configured by a manufacturer to the acquirer's requirements. These payment terminals will then be licensed or rented or more often supplied at no charge to merchants (e.g., retail stores, garages, restaurants). Multiple account acquirers may require access to their customers accounts via a single payment terminal. That is, one particular merchant may operate payment terminals which provide access to customers accounts at other account acquirers (e.g., other banks). Because of different requirements of different acquirers for handling of remote payment transactions, the payment terminal must be arranged to operate to satisfy the different requirements.

The terminal owner (often a principle acquirer) will have the terminal appropriately arranged and programmed by the terminal manufacturer to satisfy the requirements of all account acquirers utilising the terminal. Payment terminals may need to contain several programs and select the appropriate program depending on the card to be processed or on an operator selection.

It is often the case that the terminal owner may need to have the operation of the payment device amended to, for example, enable it to operate for an additional account acquirer, or to satisfy changed requirements for a particular account acquirer. Because of the different hardware/software architectures available, any operational alterations generally the require the input of the terminal supplier or manufacturer. The supplier/manufacturer will be required to reprogram the terminal or amend the hardware in order to carry out the alterations and they will usually be the only person who has the appropriate knowledge. The terminal owner is thus tied to the particular supplier/manufacturer of the particular brand of payment terminal.

It is often the case that, the terminal owner may over time obtain different brands from different manufacturers and for operational alterations may need to return the particular brand to each separate manufacturer. Over time, manufacturers may go out of business, in which case the payment terminals made by that particular manufacturer may be unsupported and any alteration may be difficult to achieve, or at least will require the input of a skilled person having detailed knowledge of the programming and/or hardware of the redundant manufacturer's devices.

Being tied to a particular manufacturer for a particular brand therefore causes cost, time and trouble when any operational alterations are required. There is therefore a

reluctance to carry out operational alterations, which sometimes means that requirements of various account acquirers are not fully satisfied. When an operational alteration does have to be carried out, it is costly. If a manufacturer goes out of business, the terminal owner may be left with nobody to alter the operation of his payment terminals, or indeed maintain the payment terminals. The present system is costly and inflexible.

A payment terminal device usually includes a microprocessor and a number of peripheral units (e.g., card reader, display, printer, communications interface, _etc) controlled by the processor. A payment terminal device usually comprises hardware, an operating system or a BIOS and is ready to accept an application for that arrangement. Or the device may be supplied with an interpreter to accept the applications.

To alter the operation of payment terminals, a new application must be created. This can be time consuming, costly and as the programming will be different for different types of devices, which may have different hardware arrangements as well, and must be carried out separately for each different type of device (i.e., different reprogramming operations must be carried out for different devices even where the same operational alterations may be required).

The programming alterations are not "portable" between different types of devices.

The most time critical aspects of operation of a remote payment terminal involve the building up and breaking down of "messages" and the formulation and operation of communications. By "messages" is meant, for example, information data which is required to be input to the device or communicated or displayed in order to enable carrying out of a remote payment transaction, and includes information to be communicated to the bank, e.g., customers card number, customers PIN, amount of transaction, etc; displayed information such as "Please Input Amount"; information to be read from a customers magnetic stripe card or smart card and manipulated by the device e.g., card number, expiry date, etc. The operation of payment terminals is greatly concerned with the collection, rearrangement and communication of this message data to enable a remote payment transaction.

In conventional devices, each time a message is constructed or deconstructed, the operation of the machine will be handled by the application program. To change operation of the machine, the application must be changed. This is laborious, and gives rise to problems, as discussed above.

The technique of creating a _virtual processor_ (or in this case microprocessor) is well known and referred to as an interpreter. This allows programs to operate independent of processor. With the newer technique of also creating virtual peripherals then the whole is referred to as a "virtual machine".

A_virtual machine_ is computer programmed to emulate a hypothetical computer. Different incompatible computers may be programmed to emulate the same hypothetical computer. Any computer programmed to emulate the hypothetical computer will thus be capable of executing programs for the virtual computer. This creates a complete portable environment for program operations.

A problem with virtual machines is emulation is slower than normal program execution. For some applications this performance penalty is a significant problem.

The above problems and disadvantages which have been discussed specifically in relation to devices configured to process payment transactions also would apply to devices configured to prepare and process any information to be sent or received via a network, not restricted to payment transaction information.

SUMMARY OF INVENTION

A first aspect the present invention is directed to a device arranged to process messages for communications, comprising a virtual machine means including a message processor means which is arranged to process messages communicated to and/or to be communicated from the device, and message processor instruction means, arranged to provide directions for operation of the message processor means. Another aspect of the present invention is directed to a method for operating a device arranged to process messages for communications. Another aspect of the present invention is directed to a method of programming a device arranged to process messages for communications.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

From a first aspect the present invention provides a device arranged to process messages for communications, comprising a virtual machine means including a message processor means which is arranged to process messages communicated to and/or to be communicated from the device, and message processor instruction means, arranged to provide directions for operation of the message processor means.

"Communicated" includes transport of data via a data carrier such as a smart card.

The message processor means is preferably a program module the specific function of which is to assemble, disassemble and compare messages. By messages we mean a sequence of data comprising usually a plurality of information fields to be communicated.

The message processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The message processor instructions are preferably virtual instructions to be expressed only in the language defined by the message processor means and thus never requiring translation to any real hardware processor.

The message processor means in at least a preferred embodiment provides two specific advantages over conventional arrangements

1) Faster Operation. The processor (executing as native code) operates at full microprocessor speed overcoming the problem of slow emulation speed for message related functions.

2) Faster, simpler programming. The instructions for the message processor preferably consist of actual message "descriptions". The programmer need only describe the message content, all data conversion, manipulation and processing is automatically performed based on the message description. This is a more intuitive and compartmentalised approach which preferably leads to faster programming with less errors.

The virtual machine preferably also includes a protocol processor means particularly arranged to organize communications to and from the device, and also preferably include protocol processor instructions which are arranged to provide directions for operation of the protocol processor means.

The protocol processor means is preferably a program module the specific function of which is to control and select

the sequence of message processor operations in relation to messages received and transmitted.

The protocol processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The protocol processor instructions are virtual instructions expressed only in the language defined by the protocol processor means and thus never requiring translation to any real hardware processor. The protocol processor means provides two specific advantages over conventional arrangements:

1) Faster Operation. The processor (executing as native code) preferably operates at full microprocessor speed overcoming the problem of slow emulation speed for protocol related functions.

2) Faster, simpler programming. The instructions for the protocol engine preferably consist of an actual diagram of the message flow. To change message flow or sequence, the programmer can modify an intuitive diagram, all multiprocessing and other complications are handled automatically. This more intuitive and compartmentalised approach leads to faster programming with less errors.

In a preferred embodiment, therefore, a device in accordance with the present invention includes a virtual machine including virtual processors which are specifically arranged to control message construction, deconstruction, comparison and to control the communication of information, both for reception from a network and transmission to a network. These operations can therefore be carried out at speed, overcoming the problems with known virtual machines and interpreters, which tend to operate slower than conventionally programmed devices. The virtual machine therefore lends itself particularly to applications relating to communications, such as payment terminal devices and other devices in which message processing and communication comprise a significant proportion of the operation of the device. In payment terminals, for example, a payment terminal including a virtual machine having the message processor means and protocol processor means can operate satisfactorily speedwise. The virtual machine can be implemented on any hardware, BIOS/OS arrangement and therefore facilitates portability of programs.

Implementation of such a virtual machine on payment terminal devices of different brands enables operation of the payment terminal devices or brands to be altered merely by altering application commands generic to all brands. Each brand is seen by the application as the same virtual machine.

The virtual machine preferably also includes a function processor means arranged to control overall virtual machine action in response to operator or other external events, and also preferably includes function processor instructions which are arranged to provide directions for operation of the function processor means.

The function processor means is preferably a program module the specific function of which is to control and select general operations of the device not specially controlled by the message and protocol processor means.

The function processor means is preferably translated into the native code of the microprocessor in each hardware device on which the virtual machine is to be implemented. The function processor instructions are preferably virtual instructions to be expressed only in the language defined by the function processor means and thus never requiring translation to any real hardware processor.

In the preferred embodiment, the "application" will therefore comprise instructions for the message, protocol and function processor means. The instructions for the function processor means may include such prior art modules as a function event scheduler and function selector.

Although the present invention is particularly applicable to application in payment terminals, it is not limited to such applications. The invention can be applied in any device where advantages are likely to be achieved for the arrangement and control of communications.

With the advent of the Internet and other extensive communications networks, it is believed that the operation of computers, such as PC's, will become more and more oriented towards acting as "servers" and/or "browsers". In other words, a major function of PC's connected to a network will be to operate either as a server, providing information and/or programs to the network for access by other parties, or as a "browser" for obtaining information/programs available on the network and operating on them. It is likely, in fact, that PC's will be asked to operate as both a server and a browser. This operation will not merely be restricted to the Internet, but for any network, even Local Area Networks.

The applicant also believes that many other classes of devices may be connected to a network. For example in the future a home video cassette recording machine could be connected to the Internet (along with other devices) allowing remote programming from a browser device. An example of the use of this would be a worker upon learning of a requirement to stay at the office late and miss a favorite show could access their home VCR from the office and program it.

Telephone calls will eventually be digital and most likely use the Internet as the digital network. Like the VCR, this does not mean all phones would need a qwerty keyboard and color display. They will both represent other classes of Internet connected devices not requiring the exact same configuration as PC's.

The present invention facilitates the production of a small, economical device which is particularly arranged to deal with communications, to build, compare and deconstruct message information. Such a device is novel maybe termed a Specialized Network Access Computer (SNAC). The applicants believe that a SNAC could emerge as a class of device allowing data entry and control through the Internet where a smaller, more economical device than a conventional PC is appropriate. In a preferred embodiment, the device is implemented utilising a virtual machine having a message processor and a protocol processor as discussed above. In the preferred embodiment, the software of the device can be considered to include three layers of virtual machine software (the HW drive layer, the Hardware Abstraction Layer, and the Virtual Machine Processor layer) and a software application. All layers other than the Virtual Machine Processor Layers are well established by prior art. A payment terminal can be used substantially without alteration as the hardware component of the device. A hardware abstraction layer (HAL) is a set of routines providing a common application program interface (API) to exercise the operating system, BIOS or hardware drivers.

HAL consists of routines to either (a) implement the functionality not provided by the underlying operating system, BIOS or hardware drivers, but needed for the common API, and (b) translation of parameters and adjustments of functionality required to adapted underlining OS, BIOS routines for the routines specified by the common API.

Such a SNAC can be applied in many different types of communication application over a network.

The present invention also facilitates the production of devices which incorporate a snac as a functional element of

the device. Such devices could include both devices collecting information for transmission over a network such a pay telephones, particularly those equipped with smart card facility, or devices receiving information from a network such as the futuristic VCR or even washing machine.

Preferably, message instructions and protocol instructions may be developed on a convenient device such as a PC or general purpose computer, utilising a development tool in accordance with another aspect of the invention.

From a further aspect, the present invention provides a development tool for developing message instructions for providing directions for operation of a message processor means to be implemented in a virtual machine as discussed above, the development tool comprising a processing apparatus arranged to receive data input by a user to build message instructions for the message processor means.

The arrangement is preferably driven by a graphical user interface based program which provides various screens and fields for the user to input data relating to message instructions.

The message instructions are preferably subsequently converted to code and downloaded into the device which is to employ them with the virtual machine. From a further aspect the present invention provides a development tool for developing protocol instructions for directing operation of a protocol processor means to be implemented with the virtual machine as discussed above, the development tool comprising processing means arranged to receive data input by a user to build protocol instructions.

The arrangement is preferably a program which is arranged to build protocol instructions from the data input by the user. The program is preferably graphical user interface based and provides screens and fields to facilitate data input for the protocol instructions.

Protocol instructions and message instructions can therefore be built on a PC and downloaded to device where the virtual machine is to be implemented.

A tool has also preferably been provided for developing function processor instructions, along the lines of the tool for the protocol processor instructions and message protocol instructions.

Limited hardware provided by such a device as a payment terminal or other SNAC device does not lend itself to development and testing of applications programs. Although the finalized application must run on the hardware, to develop and test an application it is more convenient to be able to utilize a more user friendly device, such as a PC or general purpose computer.

From a further aspect, the present invention provides a communications device including a virtual machine means including a protocol processor means arranged to organize communications to and from the device and protocol processor instruction means arranged to provide directions for operation of the protocol processor means.

From a further aspect, the present invention provides means for emulating a virtual machine on a PC or other general purpose computer, the virtual machine comprising a message processing means as discussed above. The virtual machine is arranged to operate on the PC or other general purpose computer so that instructions developed for the machine can be tested.

Similar emulation is preferably provided for the protocol processor means.

Emulation can therefore be used to test payment terminal or other SNAC application programs.

The present invention further provides a method of operating a communications device, comprising the step of processing messages for communications by employing a virtual machine means including a message processor means processing messages for communication to and/or communicated from a remote device, and message instruction means providing directions for operation of the message processor means.

The method preferably also includes the steps of processing communications by employing a protocol processor means and protocol processor instructions providing directions for operation of the protocol processor means.

Message processor means, message instructions, protocol processor means and protocol instructions are preferably as discussed above in relation to previous aspects of the invention.

The present invention yet further provides a method of programming a device for processing communications, comprising the steps of loading a processing means of the processor means which is arranged to process messages communicated to and/or to be communicated from the device, and message processor instruction means arranged to provide directions for operation of the message processor means.

The method of programming preferably also includes the step of loading the processor means of the device with a protocol processor means arranged to organize communications to and from the device, and protocol processor instructions arranged to provide directions for operation of the protocol processor means.

The present invention yet further provides a computer readable memory storing code for implementing a virtual machine comprising a message processor means arranged to process messages communicated to and/or from the device.

From yet a further aspect the present invention provides a computer readable memory storing code for implementing message processor instruction means arranged to provide directions for operation of a message processor in a virtual machine means, the message processor being arranged to process messages for communication to and/or from a device.

From yet a further aspect the present invention provides a computer readable memory storing code for implementing the virtual machine including a protocol processor means arranged to organize communications to and from a device.

From yet a further aspect the present invention provides a computer readable memory storing code for implementing protocol processor instructions arranged to provide directions for operation of a protocol processor means arranged to organize communications to and from a device.

From yet a further aspect the present invention provides a specialized network access computer, including a micro processor and a virtual machine means, the virtual machine means including instructions for running on a virtual micro processor and an interface enabling the micro processor to operate the virtual processor.

Preferably the specialized network access computer is a payment terminal or other type of "card computer" (being a computer which is arranged to process information from cards and/or communicate information to cards—cards being smart cards, magnetic cards or similar).

The interface between the actual processor and the virtual processor preferably includes a hardware abstraction layer (AJL) or the like which provides a common API.

Features and advantages of the present invention will become apparent from the following description of an embodiment thereof, by way of example only, with reference to the accompanying drawings, in which:

FIG. 1 is a schematic block diagram of a payment terminal in accordance with an embodiment of the present invention;

FIG. 2 is a schematic diagram of a control program architecture for the embodiment of FIG. 1;

FIG. 3 is a schematic flow diagram illustrating device operation which requires the operation of the message engine;

FIG. 4 is a schematic flow diagram illustrating an example of operation of the protocol engine;

FIG. 5 is a representation of a display (screen dump) available on a development tool for developing a program for a device in accordance with an embodiment of the present invention, illustrating development of a message instruction for an example message;

FIG. 6 is a screen dump of a further development tool display illustrating further detail of development of a message instruction;

FIG. 7 is a further screen dump of a development tool display illustrating further detail of development of a message instruction;

FIG. 8 is a screen dump of a further development tool display illustrating development of a further message instruction.

FIG. 9 is a screen dump of a further development tool display illustrating development of a protocol instruction;

FIG. 10 is a screen dump of a further development tool display illustrates further detail of development of a protocol instruction;

FIG. 11 is a schematic diagram showing a structural embodiment of the message instructions and description for the message processing means, and

FIG. 12A is a schematic diagram showing the structure of protocol instructions for an embodiment of the protocol processor means.

FIG. 12B is a representation of a display of a development tool for developing protocol instructions.

An embodiment of the invention will now be described particularly with reference to a payment terminal device. The invention is not limited to payment terminal devices and the following description is given as an illustrative example only. The invention can be employed in all devices concerned with communications over a network, such as a Specialized Network Access Device.

A payment terminal device in accordance with an embodiment of the present invention is illustrated in FIG. 1. The device hardware comprises a processing means which, in this embodiment includes a central processing unit 1 and a memory 2 for storing instructions and data. The device further comprises a keyboard 3 for input; a card reader for inputting information from a card 5; a display 6; a printer 7, and a communications interface 8 for communication with an account acquirer.

Prior art devices generally have similar arrangements to that illustrated in FIG. 1. The number and type of peripherals to the CPU may vary, but the essential operation required by the prior art and the present invention are similar.

Such devices operate to facilitate remote payment transactions, and a general overview of operation is as follows:

(1) Information is taken from an account holder's (customer) card 5 via a card reader 4. Transaction information is input via the keyboard 3. The transaction information may include a money amount. The display 6 may prompt the user (merchant employee, customer) to input information (e.g., it may ask a merchant employee to input an amount) and may also display information as it is input. The keyboard 3 may also be used by the customer to input a code for the account, such as a PIN number.

(2) The CPU communicates the information via communications interface 8 with an account acquirer computer. The account acquirer computer may carry out a transaction (e.g., deduct money from the customers account and pay the merchants account) or may provide an "authorization" that a transaction can be carried out. Information that an account transaction has taken place or that the account acquirer authorizes a transaction to take place is transmitted to the communications interface 8 from the account acquirer computer. A display 6 may be provided to indicate that the transaction has occurred or may proceed.

(3) When the transaction is complete, a print out of transaction information may be provided from printer 7.

Prior art payment terminal devices are generally programmed in a conventional manner. That is, programming comprises a sequential set of operating instructions which are executed in sequence to carry out a remote payment transaction. This "sequential program" may be directly compiled onto the processor of the device so that the device is under direct program control or, as is more usual, an applications program in a conventional programming language may control operations through a BIOS/OS. Whatever conventional programming form is used, however, the device suffers from the problems which are discussed in the preamble of this specification. The programs are not portable between devices having different hardware or operating system architectures and it is necessary to write a program specifically for each type of device. Further, any amendments to the operation of the device must be programmed by a programmer having knowledge of that particular device and program arrangement.

FIG. 2 is a schematic block diagram illustrating architecture of a device in accordance with an embodiment of the present invention.

The architecture comprises the hardware 100 the device, as illustrated and described in relation to FIG. 1. It also comprises the hardware drivers, known in the prior art, and including an existing BIOS/OS or hardware drivers, reference numeral 101 and also includes the Hardware Abstraction Layer Interface (HAL) 102. The HAL 102 and hardware drivers 101 form a layer of a virtual machine which also includes virtual machine processors 103.

The virtual machine 101, 102, 103 is arranged to emulate a hypothetical payment terminal. Application 104 controls the virtual machine 101, 102, 103 which in turn controls operation of the hardware 100. The virtual machine 101, 102, 103 can be adapted for many different hardware 100 arrangements (i.e. many different brands of payment terminal). Different arrangements of hardware 100 can therefore be controlled by the same application software 104.

The provision of Hardware Abstraction Layers and hardware drivers for virtual machines is known in the prior art and fully described in various publications. Each peripheral of the virtual machine is defined to be able to act in some manner on a standard set of commands. The HAL implements the best interpretation of each command on the actual peripheral present. For example a printer is defined to implement a "feed paper ready for tear off" instruction. On differing roll paper printers this requires feeding a different number of lines, on tractor feed printers this requires feed to the next perforation.

The virtual machine processors include a message processor **105** and a protocol processor **106**, implemented in software code. The message processor is arranged to process messages communicated to or to be communicated from the payment terminal via the communications interface **8**. The protocol processor is arranged to organize communications to and from the device, and to control and select the sequence of message processor operations in relation to messages received and transmitted. The message processor **105** and protocol processor **106** are implemented in native code of the payment terminal and therefore operate at relatively high speed. Because much of the "work" of the payment terminal is in building, comparing and deconstructing messages and processing communications, the operation of the device is relatively quick even though employing a virtual machine, **101, 102, 103**.

The virtual machine processors **103** also comprise a function processor **107** the operation of which is to control and select general operations of the device not specially controlled by the message and protocol processors **105, 106**. The function processor is also preferably implemented in the native code of the microprocessor of the hardware **100**. The application **104** includes protocol instructions **108**, message instructions, **109**, function support **110** and function instructions **111**. The protocol instructions **108** govern operation of the protocol processor **106**. The message instructions **109** provide directions for operation of the message processor **105**. Function support **110** and function instructions **111** govern operation of the function processor **107**. The application **104** and virtual machine **101, 102, 103** operate on data **112** input to the payment terminal to process it in accordance with the application **104**.

In this example, the application include a set of "primitives" which are a series of symbolic commands which are executed by the device to control carrying out of a remote payment transaction. The appendix A to this specification lists primitives utilized by a preferred embodiment of the invention and gives descriptions of their respective functions. It will be appreciated, however, that a skilled person would be able to design their own primitives for carrying out remote payment transactions and the invention should therefore not be considered limited to use of the primitives listed in the appendix. It is in fact anticipated that users of the system may desired to created their own primitives and product documentation attached includes instruction for this procedure should it be desired.

Appendix A is in the form of a "HELP" file to be used with a product. The important information for the purpose of this description is the brief description of each "PRIMITIVE" and their function.

The primitives operate utilising the data **112**. The data **112** may be data being input to the device, such as the customers account number, information which is fixed (strings) in the device e.g., a particular account acquirers identity.

The function processor **107** includes an event scheduler and index as known in the prior art. In response to an event (e.g., swipe card) the event scheduler operates via the index to look up a sequence of primitives **11** to be executed in response to that particular event.

In the preferred embodiment, the virtual machine processors **103** are constructed using C and the application is constructed using C++ or Java.

The device of this embodiment is event driven. When converting a device incorporation the SNAC hardware requirements to a SNAC by the provision of an appropriate HAL and virtual processors, and event driven structure can be added to a non-event driven underlying architecture through the HAL. This can be achieved through a software loop detecting events and generating an event call for any detected event.

The application **104** responds to the occurrence of an event to dictate subsequent operation of the device. For example, when a card is swiped through card reader **4**, the appropriate sequence of instructions from the application **104** will be implemented. The event driven structure allows the hardware drives **101** to have control during idle periods. When an input event occurs the application is called to process the input and then returns control to the hardware drives **101**.

The application may be loaded on a remote payment terminal device with a pre-existing operating system. Where the operating system is event driven HW drivers **102** can operate as an interface layer without any problems. Where the pre-existing operating system (HW drivers) is not event driven, amendments must be made via the HAL to convert to an event driven structure.

Appendix B includes a description of a operation of the HAL **18** in accordance with an embodiment of the present invention, on a functional level. A skilled person would be able to develop an appropriate HAL structure for an existing device or a new device. The appendix B is in the form of a "HELP" file for a product. It merely describes an example of implementation of a HAL and adaptation of an existing devices existing BIOS.

FIG. **3** illustrates an example of an operation of the device, for one typical step in a remote payment transaction. The other steps in the remote payment transaction are carried out in a similar way. That is, they may require the operation of the message processor **105**. They are event driven, such that the application **104** is called up to deal with any particular event after the event occurs, etc.

The operation schematically outlined in FIG. **3** is that of reading information from a customers card and storing information in fields for subsequent processing by the application **105**. In overall operation of the device, the information from the card will be required to identify a user and enable access to the user account to cause a transaction or authorize a transaction.

FIG. **3A** illustrates example information included on a magnetic stripe on a magnetic stripe card **5**. The information includes track 1 information, track 2 information, track 3 information, the customer name, the PAN, the expiry date and End-Of-Form label. This information must be taken off the card and stored in appropriately labelled fields so that it can be accessed to enable processing of the transaction.

At step (1), on a card swipe of card **5** through reader **4**, the card swipe event is detected by the HW drivers **101**.

The HW drivers **101** causes a call back to an event table in HAL **102** for the peripheral card reader **4** which contains a series of names for routines to be performed on the occurrence of a particular event on the card reader **4**. There are also event tables for the other device peripherals.

FIG. **3B** is a schematic illustration of the event table for the card reader **4**. Event "2" is for card swipe. In this example, there are three alternatives available for a card swipe event, labelled "1", "2" and "3". These labels may be dynamically updated in the event table, depending upon the particular stage of operation of the device.

Label "1" is for the routine "handle idle card". This is a routine which is instigated where no payment transaction routine has yet been instigated, i.e., this is "kicking off" operation.

Label "2" is the label for the "handle card" routine. This is where the payment terminal device is waiting for a card

read event, e.g., where one has a device of the type which requires a money amount to be input before the card is read.

Label "3" is where the device may be at a stage in the operation where it does not require a card reader, i.e., the card is swiped in error. In this case, nothing happens and no routine is initiated.

Note that the above descriptions of the routines are not "primitives" but are merely general descriptions.

It will be appreciated that the event table may contain labels for any number of events to carry out operation of the device peripheral the card reader 4. Similarly the other event tables for the other peripherals will be configured with labels for various routines they are required to carry out, as will be appreciated by the skilled person. It is not necessary to go into detail detailing all the routines, as they will vary from device to device and will be a matter of choice of the skilled programmer, and the operator of the payment terminal device.

This event table driven structure is ideal. In a conventional terminal, where the terminal is executing sequential program instructions, for "handle card" routine the device will merely sit in a loop waiting for a card to swipe. With this architecture, however, the device does not have to sit in a loop waiting for a card swipe. It can leave the application program and return to the HW drivers 101 and in the meantime the CPU 1 can be carrying out other jobs.

With the event label, the sequence of the application instructions for the particular routine is then looked up via an index from the application 104. The function processor 107 is then called up, step (3) to commence implementation of the instructions for card swipe. The function processor 103 then implements the instruction sequentially. The function processor 103 is a conventional interpreter, as will be understood by those skilled in the art, arranged to implement the high level primitives of the application 104 via HW drivers 101.

The first primitive requiring execution for the "handle card" routine in this example is the SAVE primitive, step (4). The first operation of the SAVE primitive is to call up the message processor 105. The message processor 105 is a series of several subroutines implemented in the native code of the CPU 1, the specific operation of which is to construct, deconstruct and compare messages in accordance with message instructions 109 from the application 104. The SAVE primitive will have associated with it a label indicating the particular message instruction 109 associated with this particular event. The function processor 107 fetches the message instruction 109 for this event and the message processor 105 then operates to load the data from the card into labelled fields (steps 5, 6 and 7) according to the message instructions.

Once the message processor 105 has loaded the information from the card into the appropriate fields, in accordance with the message instructions 109, the SAVE function is completed and the device proceeds to carry out the next function in the sequence for "card swipe" fetched by the function processor 107. Alternatively, the sequence of functions for "card swipe" may be completed and the device may wait for the next event before proceeding further.

There are a number of ways that the payment transaction could continue once the SAVE function has been carried out. For example, steps could be taken to create a display asking the customer to input a PIN. Again, such steps would be carried out by the function processor 105 implementing the instructions, which would include a function to call up the message processor 105 to build a "form" to display the request on the screen. Alternatively, the device could be controlled to take steps with regard to the information loaded into the fields by the card in accordance with the SAVE function. For example, it could compare a PAN number taken from the card with an equivalent PAN number stored in memory of the device to establish the identity of the account acquirer. A skilled person will realize that a number of possibilities are available for continuing with the transaction, and would be able to formulate appropriate programming from this description and the following appendices.

As discussed, the message virtual processor means is directed by message instructions 109.

FIG. 11 is a schematic diagram illustrating the structure of the message instruction means 109. The message instruction means is in fact in the form of a set of "descriptions" of the messages. Each message usually comprises a plurality of fields 120, and the message instruction means for each message contains a corresponding plurality of message instructions. One field may be the CUSTOMER NAME, for example. In the message instruction means, each field is associated with a number of message descriptors 121 which designate characteristic to be applied to the information in that field or to be expected of the information in that field. Operations which may be carried out on the data included in that field may also be included in the descriptors 121. As illustrated in the drawing, the descriptors may include:

1. Data Location Identification. This will indicate either where the data is to be found and/or where data is to be put. In the current embodiment the data location information is contained in a two byte field descriptor (thus having 65535 different possible values) with value ranges allocated to

1) 2000 strings

2) literal numeric values from 0 to 32,000 in abbreviated form

3) data field Ids where each ID is represented as an entry in a table, and each table may contain up to 256 fields.

2. Data Representation (i.e. Ascci, Binary, etc.). This indicates what representation form the data is in and/or what it is to be converted to.

3. Format. This provides a description of the format that the data is in and/or is to be placed in.

4. Test Function. The index of a function processor set of instructions to determine if the current field is to be included or excluded at this time

5. Line & Column. Relative position for use in constructing messages for display or printing. These values are used to determine the quantity of space characters, and or new line characters that are required in the buffer.

6. Substitution list. A list of text representations to substitute for numeric values e.g., display the value "1" as "Monday" and "2" as "Wednesday".

7. Additional description options as required by the application or prove useful in future embodiments.

Each message instruction will therefore include a description of a field of message data, providing instruction for the virtual message processor means which enable it to carry out a number of tasks:

1. To compare a message with a message description to see if it is the correct required message.

2. To take a message of the correct description from a location and place it in an-other location.

3. To take a message and deconstruct it into various components and place the various components into other locations.

4. To take data and build a message in accordance with the message description and place the built message in a location.

5. Compare one message with another message.

Other functions may also be carried out by the message processor as required by the application. The message processor can manipulate data in any desired way in accordance with descriptions provided by the message instructions. Messages comprising data can therefore be billed, placed in locations, taken from locations, deconstructed with elements being placed in locations, etc. for subsequent operation on the data by the application. Any device which deals with significant amounts of messages in such form can therefore benefit from this arrangement.

Each message description is labelled so that it can be identified by the application, e.g. each message description may be numerically labelled.

A development tool for developing the application **104**, in particular the message and protocol instructions **108**, **109** comprises a graphical user interface based program which may be run on a PC or other general purpose computer. The program provides a graphical user interface based framework which enables message instructions to be built from data input by a programmer. Message instructions can subsequently be translated into code readable by the virtual machine **102**, **101**, **103** and downloaded into the application device. FIGS. **5**, **6**, and **7** are "screen dumps" which illustrate displays generated by the development tool for an example message instruction. In this case the message relates to data from a magnetic stripe of a customers card. The message instructions direct the message processor **105** to take the fields of the message and place them in known locations in accordance with the instructions. Such a message instruction may be called up in response to the SAVE primitive, in the event of a card read. Data from the magnetic stripe of the card would be stored away in the appropriate locations in accordance with the instructions, for subsequent processing.

Each message is provided with a message name **30**, in this case "TrackData". This message name identifier can be used to call up this particular set of message instructions in the development tool. An alternative numeric identifier is generated for use by the virtual processor. This numeric identifier may also be displayed by the development tool. Each message is made up of a number of message "fields" **120**. In this particular examples there are seven fields, being "Track1", "Track2", "Track3", "CustName", "PAN", "ExpD" and "End-Of-Form". Each of the seven field is converted to a message instruction for use by the virtual message processor. This is the information which is typically found on any magnetic stripe card. The message instructions in accordance with this embodiment direct the message processor to process these elements. Each field is associated with descriptors which provide further instructions for the handling of that element. FIG. **6** illustrates a display **33** which enables a programmer to provide message descriptors to CustName element.

Each field **120** has a "format" descriptor **34**. There is an instruction as to the Data Representation ("Type") reference numeral **35**. In the illustrated embodiment there are four types, Ascll, Hex, Binary and BCD. There is also a logical operation instruction (option test), reference numeral **36**. This logic instruction can be used to determine whether or not the message processor will process this element at all, for example, i.e., it will only include the CustName element in the message when the logic function equals "True". Other instructions designate the data source, reference numeral **37**, in this case a field, and the field label, reference numeral **39**. The format **34** is labelled with a name, in this case, "Tracks". There are further instructions which dictate the format Tracks to be applied to CustName. FIG. **7** shows a display which illustrates the instructions for the format "Tracks".

The message processor is responsive to all the message instructions to load the data from the magnetic stripe card into the appropriate fields with the appropriate formats in accordance with all the rules designated in the instructions.

This embodiment of the present invention includes another class of message instruction means, known as a "Form". Instead of a Data Representation as a message descriptor, a Form includes description of a Location of the data field in the Form. FIG. **8** is a display provided by a development tool enabling the programmer to prepare message instructions for a Form message. On the left hand side of the display a panel **70** illustrates Form layout. The fields in the Form include MerName, Address Line 1, etc. The location of these fields can be moved within the panel **70**. The location in the panel is provided as a descriptor and for the message instruction. The Form type of message instruction controls displays, reports, printouts, and the like. The type of Form is given by the instruction designated by reference numeral **71**, in the example illustrated in FIG. **8** being a printout. The message processor takes the fields from known memory locations or other locations and enters them in the locations enabling the Form described by the Form instruction to be produced.

As discussed previously, another major function of a SNAC device is communications. For example, it is necessary for the majority of remote payment transactions for communications to be able to occur between an account acquirer location, in order to enable access to an account, and the remote payment device. Communication with a data carrier, such as a smart card device may also be required.

The protocol processor **106** is arranged to organize communications, in accordance with directions from the protocol processor instructions **108**. Referring to FIG. **4**, in a typical remote payment transaction, after a card has been swiped, a PIN number has been input and a charge amount has been input, information then needs to be communicated to an external computer, at the account acquirers, in order to enable further processing of the transaction. After an event such as a communications message arriving, therefore, HAL **102** detects the event (step 1) and activates the protocol processor (step 2), FIG. **4**. The protocol instruction **108** for the event is rolled up (step 3). The protocol processor **108** implements the protocol instructions for that event, (step (4)).

The protocol instructions are divided into "sections" **130**, "lines" **131** and "protocol commands" **132**, as illustrated in FIG. **12**A. FIG. **12**B illustrates how an instruction is displayed on a development tool for protocol instructions. Protocol instructions describe message flow both from and to the device. The top line specifies outgoing messages and the other lines display possible incoming results. A protocol consists of lines and sections. At the start of each section is a line 1 (optional for the first section) which describes the outgoing message. There are a number of protocol commands, and these include:

1. Protocol—Run a sub protocol

2. Message—Send a message or handle an incoming message using the virtual message processor means

3. Retry—re execute the steps of protocol from and indicated point

4. End —End of the protocol

5. Exit—Stop the protocol from an intermediate point

6. Timeout()—Specify the a delay after which the protocol should automatically jump to the point at which the timeout instruction is placed.

7. Control—Specifies a control character to be send or received.

8. Function—Execute a virtual function processor function

Protocol instructions are organized in lines and sections. In each section Line 1 indicates the information to be send by the SNAC device and subsequent lines indicate actions to be taken in response to the alternate possible events which may occur in reply. The first instruction on each of these subsequent lines is used to identify the response. Control( ), Message( ), Function and timeout( ) may all be used to identify responses as follows.

1) When the time specified by a timeout instruction elapses then the line commencing with the timeout will be selected.

2) When data is received it will be sequentially compared to a lines commencing with Control( ) Message( ) or Function to see if the data matches the control character, matches the message of causes the test contained in the function to evaluate to true.

FIGS. 9 and 10 illustrate displays of a development tool for protocol instructions for the protocol "General" which is the Protocol Name (reference numeral 42). Instructions are presented as a screen dump in the form of a table 43, which can be accessed by a programmer if he wishes to alter the protocol.

Protocols are arranged to control message flow both from and to the target device (e.g., account acquirer computer). The top line of the display panel 44 specifies outgoing messages and the other lines display possible incoming results.

A particular protocol is able to call up other protocols "nested" within it and is also able to call up the message engine to deal with messages.

Referring to FIG. 10, the top line of panel 44 specifies the outgoing message. The first operation of protocol "General" is to call up and carry out a further protocol, "Reversal". FIG. 11 illustrates instructions for the protocol Reversal, reference numeral 45. Reversal operates to call up the message engine to construct message number 0400 and this message is then sent to the target device.

The either

1). Message number 0410 should then be received back from the target device and the message processor will be called up to deal with that data, which involves the message processor comparing the incoming message against the description specified by the message instruction means and storing the data if a match occurs. Or

2) A timeout of 100 tenths of a second elapses.

Then the protocol is ended and re-turned to the protocol General, which causes a further message, 0100 to be formulated and sent out.

Then either

1) A message matching 0110 will be received or

2) A message matching 820 will be received or

3) neither 1 or 2 will occur for the timeout( ) period, in this case specified as 000 tenths of a second.

If the message 0110 should then be received from the target device and compared by the message engine, then another protocol "adjustments" will then be carried out. The protocol would then end.

If the message 820 should be received from the target device, which can be dealt with by the message engine and compared with the instructions from the message instruction means. The "Retry" instruction will then be executed causing the virtual protocol processor to move execution back to the sending of the (0100) message. The retry count of zero indicates this loop would continue whilst 820 messages are received.

If the Timeout occurs, then the retry(5) would be applied causing the protocol processor to move execution back to the Send 0100 message. This loop would occur up to five times as indicated by the retry(5). After the fifth time execution would move to the next section causing the protocol to End.

More details of operation and build up of messages and protocols are given in the appendix A.

The device in accordance with the present invention for example a payment terminal, may be implemented in GAVA by defining a class library payment terminals. This class library would contain calls to all the functions of how to HAL and preferably the message and protocol engines. Similarly, a specialized network access computer or card computer could be implemented in GAVA.

Please note that the arrangement of the present invention can be used to deal with any payment transaction device, including one which deals with smart cards.

The present invention can also be used to implement a specialized network access device, which may use similar hardware to that provided for a payment terminal.

In the attached Appendix A, the term "CardScript" is the name the applicants have given to programming required to implement this embodiment of the invention.

It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the invention as shown in the specific embodiments without departing from the spirit or scope of the invention as broadly described. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.

APPENDIX A

Contents.

Introduction

Introduction

Help for CardScript Scribe

The Scribe program assist in the design of stored information & programs for EFTPOS terminals, PINpads, Electronic Cash Reqisters and other small computer systems.

Writing a Program

For help on writing a CardScript, program, rather than operation of the Scribe tool, see

Writing a CardScript Program

A CardScript program is more similar to a Windows RAD tool program than a conventional C Language or Assember program.

The "target" device has several keys, one or more card readers, and usually one or more communications ports. Defining a program consists of attaching actions to these events, or the special events of terminal power on and terminal idle.

CardScript programs—as all other program—manipulate data. Data is defined in a Data Dictionary. Unlike normal programs, it is normal to write many CardScript programs using the same Data Dictionary. Standard Data Dictionaries are available from CardSoft for EFTPOS and several other application types. It is recommended to write initial applications based on one of these standard dictionaries. Once the program is experienced, the Data Dictionary for an Application may be modified. see

Configure Data Dictionary

Data Dictionary Usage

The Data Dictionary represents the list of all "variables" or information values used in the target device. These "variables" are in formation which may change over time, or be different from device to device.

Information which is fixed for all devices usually is defined by strings. All information to be included in displays, receipts, messages etc, comes from either the Data Dictionary or from STRINGS.

Data Dictionary Fields may Have an Initial Value Set From the Initial Data Tables

Structure

Tables

The Data Dictionary is divided into tables. Each record displayed in Configure Data

Dictionary describes one table. Fields are placed by selecting add and clicking on the Panel.

Field Attributes

Double Clicking on any field reveals and allows viewing and/or editing of Data Dictionary Field Attributes.

Field Order

Layouts are stored indexing fields by table#1field#. This means existing scripts will behave strangely if the Data Dictionary is changes the number Of referenced fields.

For example if "Merchant Name" is table 3/field 2 and "Address" is table 3/field 3. Then deleting field is table3/field1 will make any prior references "Merchant Name" now reference "Address". This can be remedied by inserting a dummy table3/field 1 as a placeholder. Generally this problem does not arise since new dictionaries are not to be used for old applications, and existing dictionaries are usually only extend. In the rare event that a dictionary used by existing applications is to have fields deleted, it recommended to rename them to "dummy" or "unused". Be careful since any existing data in the files will be rearranged when retrieved, it will simply be move from the record into the fields in the order listed at the time. New fields added in graphic display mode are always added at the end

Reserved Settings

See Reserved Data Dictionary Settings

See Also

Data Dictionary Field Attributes

The field attributes which may be set are as follows

Type

Type refers to the format in which data is held. "X-Ref" is a special value used to indicate that another table will be referenced at run time and thus must be included in the build.

Binary Data Fields

Binary, either 1 or 2 bytes in length for Integer values in calculations, longer fields hold bit fields or keys. 250 bytes is the maximum permissible number of bytes

Maximum Integer Values

Depending on the number of bytes used to represent the Binary number, the following values are possible

1 Byte	—	O . . .	255
2 Byte	—	O . . .	65,535

-continued

3 Byte	—	O . . .	16,777,215
4 Byte	—	O . . .	4,294,967,295

Text-up to 250 bytes

BCD up to 250 bytes

Date/Time (2 Bytes for Dates, 2 or 3 bytes for Time)

see Date & Time Fields

Amount—10 bytes, internal format is target device dependent

Packed Amount—not currently used

X-Ref—Advanced use only

Flags

0=Field is fixed and never reset

1=Reserved for future use

2=Reserved—used with deleted fields

3=Field is reset when terminal is loaded

4=Field is reset at power on

5=Field is reset by idle function

Bytes

The leright of stored data in bytes

Length

Caution: When you create new data dictionary fields, make sure their leright is not zero if you want to use them, or they will be invisible.

The number of characters allocated to display the field as text

Name

The name of the field for display on receipts etc.

Table

The "refer" Initial Data File from which the field initial value is extracted. Blank if the field is extracted from the default file.

Table ldx

When "Table" is non-blank, "Table ldx" specifies the "refer" of the Initial Data Field in the default file used to indicate the record number in "Table" from which data is to be extracted. In other words the join field between the default table and the joined table.

Field

The "refer" of the Initial Data Field from which the initial value of the field is to be obtained.

Creating a New Application

As suggested above to create any application, it is recommended to copy a "template" application. Simply copy the entire template directory.

To then work with the new directory, select File/Installation and edit the data directory. (Don't forget the trailing I) Is the recommended to exit Scribe and restart. Scribe and restart.

Console/Display

The display console used with CardScript is quite sophisticated. see

The Console

CardScript can be used in a variety of devices, some of which may not support all the features described here.

Features

The CardScript Console has a number of sophisticated features

Hot Keys

Keys used to launch only one action, where the action is part of the application, are known as hot keys. Typically the action may be activated only when the terminal is idle. For further information see the "KeyBd" primitive.

In an EFTPOS application, on a terminal with Keyboard Buttons available for allocation, Hot Keys will normally be allocated to such functions as "Sale", "Adjustment", "End of Day" etc.

Hot keys normally would have their label printed on the keyboard, or on the physical button.

Multiple Field Input

On any Layout displayed on the console, several field may be selected for input. The OK key steps from one input to the next. Any soft key terminates all input.

Scrolling

The display may be scrolled, permitting a larger virtual display than the physical display. Scrolling is performed automatically by the driver in the target device. All that is needed to enable scrolling is to tell the scrolling driver what keys on the keyboard perform scrolling. The keys used to scroll are set by the

Console Primitive

Console(Command,Parameter)

The command determines which of the following console options is set.

Command I—Set Scroll Keys

The Parameter is a string of four hex values, in order key-left, key-right, key-up, key-down

The keyvalues specified are assigned to the scrolling engine within the target device. Note scrolling my not function on all CardScript "targets" and the size of the scrollable area may vary.

Command 2—Set Keymap

The parameter is a Key board map. This command is now the preferred method for setting the keyboard map. The old Keymap primitive will be obsolete in a future version. See

KeyMap

Structure of Keymaps

The "field" or String used as a keymap must be a list of 4digit hex blocks, the first two digits of each block representing the hex code of the key to be mapped and the next two digits representing the hex value to map be returned. Usually used from the startup function, using a field from the terminal groups record.

Note that the following key codes have special meaning.

08

Represents a backspace or 'CLR' code.

OA

Represents an 'OK' or 'Enter'

1B

Represents a cancel.

30 through 39

Represent the digits "0123456789"

Command 3-Keybd

The parameter is evaluated to zero, or non zero.

Upon entering idle state, the action of the keyboard is determined by the last KeyBd command. The keyboard (except cancel) will be ignored if off was specified.

This command replaces the old keybd primitive, which will be obsolete in a future version.

Command 4—Invalid Entry

The parameter is the text message to be displayed.

This command is designed to be called from an input validation function. Calling this command indicates the input is invalid, the text specialized in the parameter should be display to indicate the error.

Soft Keys

Some keyboard buttons on a target device may be used as soft keys. As opposed to Hot Keys these are buttons which may be used to initiate different actions, depending on the display present when they are pressed.

Since the principle of Soft Keys is to use the same buttons for different actions, displays must in some way indicate to the user the operation of each currently active soft key.

Soft Key Button Sets

Target devices may allocate certain buttons on the keyboard for use as soft keys. Button sets are numbered from zero. If a specified set is not available, then set zero is used. By convention:—

Set 0 keys 'O' thru '9' (the numeric keys)

Set 1 are dedicated soft keys, usually positioned directly adjacent to the display, in order that the display may be easily used to indicate their function.

Set 3 is the new standard for dedicated soft keys—hex values **81**,**82** . . . AO.

Set 3 will normally be requested on forms where numeric/text input is also required.

Set 4 is the same as set 3, but allowing use of the numeric keys if no dedicated soft keys are available. Set 4 should not be specified on screens where numeric input is also available, since this may cause a conflict.

Soft Key Action Groups

These are the groups of actions that may be offered at any given time.

In Layouts/Forms, a soft key action set may be selected for any display. Individual functions may be assigned to an action group from the Function/General Purpose Functions, in the Function activation section.

Note Action group 0 is used to indicate a function is NOT part of any group

Correlation of Actions to Buttons

If a display allows soft key Button Set 0, (Keys '0','1', '2','3' etc) and action set 'I' then when the '2' key is pressed, then soft key group '1', An Group '2' (Key '2'minus the first key in the action set equals 2), if it exists, will be activated.

The Keyboard

To control the features available, and make best use of your terminal, you can recode the keys on your keyboard using the keymap primitive. This allows you to customize the target device to allow portable and easy to operate applications.

See

Keyboard Codes

The Keyboard codes are designed to accommodate a wide variety of keyboard configurations. At any given time each key (or button) will act as one of the following key types

1 Control/Data Entry Control
2Data Entry
3Soft/Hot key function activation

Control/Data Entry Control

Some keys are required for control. Control keys should not be used for any other purpose than control, otherwise the user interface will incredibly confusing.

Minimum Requirements

All CardScript drivers should provide these key codes without any mapping required.

08 Backspace or Clr

1B Cancel

OA OK or Enter

1A Fn—For general Function selection, and for double (or tripple) zero.

Additional Options

OD or complete form (combined with tab as an alternative to OA)

09 Tab (move to next field—does not complete form)

OB Vertical Tab—Used as back tab or move to prior field.

11 (XON)DCI—Used as cursor left

12 DC2—Used as cursor right

13 DC3—Dedicated double zero

14 DC4—Dedicated Function Key (combined with DC3 as an alternative to 1 A).

Data Entry

Three levels of data entry may be available at any one time. Text, Hex,Alpha. The bios can automatically determine the available level and act accordingly.

Minimum Requirements

30.39 (Numerics)

Additional Options

A B C D E F (allowing Hex data entry)

Full 'querty' keyboard

Soft & Hot keys

Soft keys previously were recommended to be 'a"b-c' etc Now the are recommended to be hex 81 82 83 etc up to a maximum of AO allowing up to 32 dedicated Soft keys. The change in recommended values is to allow for terminals with full alphabetic keyboards. Hot Keys (When/Additional to soft keys) should be allocated from A1 . . . DO

Program Portability

Portable Programs

CardScript allows the writing of totally portable programs, it is also possible to write programs that are not very portable. Any CardScript program will "execute" on any CardScript enabled target, however the result could be of no use on the target if special hardware characteristics are required for practical operation of the program. CardScript provides a mechanism for avoiding the traps and keeping programs portable whilst still taking advantage of special hardware when available.

Keyboard Traps

The key map primitive represents a trap in that this function should never be used with a literal string, or your program won't be portable.

Processing Cards

Magnetic Cards

Automatic Processing

Automatic Magnetic Card Processing, from

Upon Card read, data from the card is placed in the Receive buffer. The format in the buffer is

Track 1 (I terminated)

Track 2 (I terminated)

Track 3 (I terminated)

Customer Name (I terminated)

PAN (I terminated)

Expiry Date (6 bytes ASCII)

If the read occurred at terminal idle, the Calculation Result is set to zero and the system event Magnetic Card Read is generated. After executing any function for the Magnetic Card Read Event, if the Calculation Result is non zero this value is used to select a function for further processing of the specific card type.

Automatic Magnetic Card Processing

Upon card swipe, the data dictionary fields

Transaction/Track2 tru Transaction/Customer

Name are filled with the card details. These are data dictionary fields Table1/Field2 thru

Table1/Field7. see Reserved Data Dictionary Settings for details.

Table 5 is then scanned to find a column matching the PAN of the swiped card. If a column in table 5 is found then the tables 3 & 4 have their columns set according to the entries for Issuer & Acquirer in Table 5.

Table 5 is set during the build to indicate the appropriate action should a card be swiped at idle. If the terminal was idle at the card swipe this function is now executed.

Typical Processing.

For standard processing, create a function as follows

store (O, CardMsg)

```
if ColFind(Pan,PanLow,PanHigh)
  ColSelect(Issuer, IssuTbl)
  ColSelect(Aquirer,AcqTbi)
  Exit(O,CardFunc)
```

Smart Cards

Two primitives are available for controlling Smart Cards.

Card(Command,Field/Value)

1A command of 1 is used to read the smart card status into the field "FieldNalue". Using a value for "FieldNalue" does achieves nothing.

2A command of 2 is used to control the power to the card. If "FieldNalue" is 1, the card is powered on, if "FieldNalue" is 0 the card is powered off.

3Select. A command of 3 is used to select which smart card reader(or plug in is currently selected. By convention, 1 is the user card(or if only one reader is present, this reader),

2 is the separate merchant card slot or the plug in, where present. "FieldNalue" contains the card number to be selected.

4A command of 4 is used to read the Smart Card Type Code

5A code of 5 performs a logical test on smart card type. If the field/value supplied matches the Smart Card Type Code of the current code, the logical true flag is set. This command is designed to be used in an "if" test

6Code 6 reads the CardEntryMode into the specified FieldNalue. See CardEntryMode 7Set Card entry mode to the value specified in FieldNalue

see Also

TPDU(Command, SendMsg,RxMsg,Status)

The TPDU primitive is used to send a command to the card. If the TxMsg is present this data is also sent to the card. If the RxMsg is present then a response is expected from the card and is stored in the RxMsg buffer.

Command

This if the actual 5 bytes TPDU to be send to the card

SendMsg

This optional parameter specifies the message used to build the data send to the card.

RxMsg

This optional parameter specifies message used to store any data returned by the card in response to the TPDU.

Status

This mandatory message specifies the location of status variables to record the status of the TPDU operation.

The TPDU primitive with Synchronous (Memory Cards) 416 Cards

Drivers for 416 Cards support the following TPDU Commands

ReadBytes

WriteBytes

EraseBytes

Present Key

see Commands for Memory Cards

The present Key uses the length indicator to select either the CardSecret Code (2 bytes) or the application erase secret code (4 bytes)

Answer to Reset, & Card Type

The 416 has a card type code of 4 and an an wet to reset of

3Bh 00 00 00 00 00

Commands For Memory Cards

ReadBytes

CL (any)

INS BO

ADDR XXXX (Byte Address an Card)

LN LL Number of bytes to read.

WfiteBytes

CL (any)

INS DO

ADDR XXXX (Byte Address an Card)

LN LL Number of bytes to write.

EraseBytes

CL (any)

INS DE

ADDR XXXX (Byte Address on Card)

LN LL Number of bytes to erase.

PresentKey

CL (any)

INS 20

ADDR XXXX (Byte Address on Card)

LN LL Length of Key.

Other Smart Card Commands

For selecting the smart card reader, and control of the reader.

For sending information to the card, and receiving information from the card.

Smart Card Type Codes

1 Async ISO type Card

2416

Asynchronous SCHLUMBERGER type EE2K

Asynchronous SCHLUMBERGER type EE4K.

Asynchronous SCHILUMBERGER type EE16K.

Asynchronous type GPM256

Asynchronous generic type 12C BUS

Asynchronous type GFM2K

9 synchronous type GFIVI4K

For coding of smart card types.

Running A CardScript Program

To run the program on the PC simulator, see

PC Simulator

There is a implementation of the CardScript Virtual machine available on the PC that not only runs your program, it also emulates the keyboard layout and other controls of any target machine.

To run the simulator—select "Build/Run Simulation of Build"

Stored Information—Data Tables

For Information on setting & changing values in the Data Tables See—

Tables Menu

CardScript includes all tools for maintaining data tables to control the setup and distribution of Data Tables required for any application.

The System Data Table

The system data table has a fixed format identical in all systems. This table contains general information and current setting for use within the scribe program.

See—

System Table Settings

Settings in the system table are used to both miscellaneous settings in the script, and options for viewing the script.

Loader Settings

Terminal For Mask Load

Not used by scribe

Default Prompt (or String) Table

The string table displayed within Scribe. Multiple string tables may be used to support multi language applications.

Peripherals

Description the peripherals of a the target system here and displays and receipts will in scribe will show guides to assist in design. These setting have no affect on program execution in target devices and may be changed at any time.

Reserved Functions

Idle State

Set this pointer to indicate a function to be executed each time the "target" becomes idle.

Abort

Normally left at <none> since applications may vary default options during execution.

Initial

This function is executed at "target" power on.

Processor

Previously used to indicate the byte order used in the "target". It is now recommended to use "Low-High (INTEL)" for all systems.

Configurable Data Tables

The data tables used by CardScript can have their names, field names, layout and even the number of tables used altered according to the current system setup.

Configure Initial Data

Initial Data Usage

The purpose of initial data tables is to provide a database of information for initial values for the data dictionary loaded into the target device.

Configuration

Structure

Each record in the configuration describes one table. Fields are placed on the Panel and dragged to the appropriate position.

Field Attributes

Double Clicking on any field reveals and allows viewing and/or editing of Initial Data Field Attributes.

Field Order

Clicking on "Graphic Display" toggles between the standard graphic view of the fields and a simple ordered list of the fields. In the ordered list mode fields may be dragged to rearrange the field order.

Be careful since any existing data in the files will be rearranged when retrieved, it will simply be move from the record into the files in the order listed at the time. New fields added in graphic display mode are always added at the end.

Reserved Initial Database Settings

Certain fields must be present in the for the build process.

File Usage

File 1—"Terminals". The name may be changed however this file is used to initialize individual target devices with the optional "NetMgr" module. No other special usage at present

File 2 "Groups"—no special considerations

File 3 "Issuers" no special considerations

File 4 "Acquirers" no special considerations

File 5 "Card Ranges"—must be used as card ranges, and must have fields "lo" "hi" and "len"

File 6 "Products" no special considerations

File 7 "Region Settings" no special considerations

File 8 "Issuer Sets" no special considerations

File 9 "Card Sets" must contain the fields "cards" as an index into card ranges

see also

Initial Data Field Attributes

Type

Flags

Current usage

0=Place label to Left

1=Place label above

Repeat

The number of times the field is to appear on the form

X-Pos

The current value of X-position of the field on the form. Usually modified by dragging the field.

Y-pos

The current value of Y-position of the field on the form. Usually modified by dragging the field.

Label

The label to appear for the field on screen.

Refer

The "refer" label used to access the field when building the initial data dictionary Display (Type=Text only) The number of characters to be displayed on screen. 0 (zero) for default.

Size

Functions

For information on defining functions in your application see—

Functions Menu

see also Function Primitives

For Describing any function within the "target" to the system, or in program terms for writing scripts see

General Purpose Functions

Use this selection for describing functions

Label

The function name

Description

A brief description of the function. The function can be located by description

Action

A window to the function actions. Double click on this window to see or edit the full Function Action. see also Function Primitives & Function Primitive Categories. see

Function Action
Double clicking on a function action block brings a panel into view for editing the function actions.

Adding Actions
Select the appropriate action from the alphabetical list beside the add option, select add and then click on the panel at the appropriate position. Clicking over an existing action will insert the new action before the existing action

Deleting, Actions
Select delete and click on the action. Take care to deslect delete before clicking on other actions.

Editing Actions
Double Click on any action to activate the edit dialog box.

Function Index
Shown for reference purposes. Cannot be changed.

Strings
See your driver information. Currently this information is not used by most drivers.

Function Activation
Specifies when this function will be executed. see

Starting A Function

Function Number
Each function may be assigned a number. The operator may then enter the number and the program easily select from the list using the Function# primitive.

Hot Key Code
Each function may be assigned a hot key code. Enter a non-zero code in HEX and if a key with this code is pressed at idle, or any other time hot keys are activated, the function will be activated. Note that the Cancel Key is regarded as as system event.

System Events
System Events are similar to hot keys, only instead of keys being pressed (Note that the Cancel key, IS a system event), other actions on the target device are involved. For each target machine a list of System events is maintained, but these should always include the standard events. Only one function may be assigned to any System Event.

See

Standard Event Codes

Keyboard.
Keys on the keyboard with a value less than 128 (Ox8O hexadecimal) generate an event code with the value of the key.

Other Event Codes

0=Reserved

1 System becomes Idle

2Cancel Key Pressed

3System Power On

4Numberic Entry

5Smart Card Insertion

6Smart Card Extraction

7Magnetic Card Swiped

8Checksum Error Detected on Batch

9Checksum Error Detected on Data

Card Set
Select a card set. When any card belonging to this set is swiped at idle, the function will be activated.

Usage Flags

Operator Function—The operator of the target device selects the function

Library Function—The function is an internal "subroutine"

Not Used—The function is not used

For adding actions to functions which may be varied by issuer or by acquirer see

Transaction Function Input

Transaction Function Approval

Function Primitive Categories.
Function script is a sequence of calls to system and user primitives. For information ol

primitives available see

Function Primitives

Assignment Primitive

Field1:=String/FfeW

Set field1 to the string/field2.

=> (goes to) primitive

=> field
The value of the last logical or other operation using the "calculation result" is stored in field

specified

eg

Account ==000

=> zAccount
Would set the field zAccount to 1 if Account was zero, othersize zAccount would be zero.

=> result (goes to) primitive

=> field
The value of the last logical or other operation using the "calculation result" is stored in field specified

eg

Account ==000

=> zAccount
Would set the field zAccount to 1 if Account was zero, othersize zAccount would be zero.

see also

Calculation Result.
Calculations generate a "Calculation Result". Think of this value as the value you would see on the display of a calculator if the calculation was performed on a calculator.

Math Primitives

Field1+=Number/Field2

Field1-=Number/Field2

Field1*=Number/Field2

Field1/=Number/Field2

Field1 is modified by thefield2/number.

Relational Primitives

Field1value1<Field/value2

Field1value1>Field/value2

Field1value1==Field/value2

The two fields or values are compared. If one field is text and the other numeric then the return value will always be false.

<> !=>=<=

For not equals (whether thought of as<> or!=), greater than or equals (>=) and less than or equals (<=) use the opposite case. With the WHILE PRIMITIVE and REPEAT UNTIL primitive then use the NOT option. With the IF PRIMITIVE use the ELSE clause.

Abort Primitive

abort

This primitive causes the target device to stop all functions and become idle

Alarm Primitive

Alarm (noise type)

Makes the sound specified

1 error alarm

2bip type noise

3most severe alarm

Bit Manipulation

Bit Numbering

All binary fields can be accessed as a number of bits where the number of bits no.of.bytes*8

The MSB of each byte has the highest bit number and the LSB the lowest bit number. Note ISO bitmaps do NOT follow this rule, but these do not need bit manipulation by the application.

This result in the LSB of the last byte being bit 0 (zero) and the MSB of the first byte being no.of.bytes*8-1. Eg for two bytes 15.

Bitcount Primitive

bitcount(field,start,end,bitvalue)

start & end

These are both bit numbers. see bitnumbering. Direction of counting is from start to end, either may be the larger number.

bitvalue

0 indicates count zeros

1 Indicates count ones

2 Indicates counts zeros and stop at the first non zero bit

3 Indicates count ones and stop at the first bit no set to one.

Notes

The number of sequential bits of the value "bitvalue" starting from bit "start" and working towards "end" is counted.

If the result is non-zero the logical true status is set, otherwise the logical false value is set, allowing "if" type tests of the result

The count is stored as the "working value" allowing storage via the "->" (goes to) primitive. see -> (goes to) primitive Setbits Primitive

Setbits(field,startbit,endbit,value)

Bits number "start bit" thru "endbit" are set to the value "value". No all values are extracted from the least significant bits of value.e.g. For a 1 bit field, all values are considered either 1 or 0. (Odd numbers are 1)

Batch Primitive

Batch (Operation)

Operation 0=reset to first txn in batch

Operation 1=find & restore next txn

Operation 2=delete current transaction

Operation 3=delete all transactions

CardRead Primitive

Card (string, field form, default)

This primitive is identical in operation to the show primitive, with three extra facilities

1input is terminated by either the introduction of a smart card, or the swiping of a magnetic card.

2Data from a magnetic card read is stored in the reserved fields

3If the (card entry mode) is non zero, this primitive does nothing. This allows logic to read a card only if the card is not already read.

See also

EXAMPLE

A function requiring input of both "Tip Amount" and "Cash Out Amount" can input both using the same field.

Create a form as follows. Edit the PFIELD to indicate an input field.

PSTRING Name: Form

PFIELD

Create a function

Show(Tip, TipAmt, Form, 0)

Show(Cash, CashAmt, Form, 0)

Where "Tip" and "Cash" are strings. On the first call to show the display will prompt "Tip" and accept input into the TipAmt field. On the second call to Show the display will prompt "Cash" and accept input into the CashAmt field.

Show (string, field form, default)

Action

This primitive is specialized for displaying input forms. Two parameters (string and field) substitute with PSTRING and PFIELD in forms, allowing the same form to be used for multiple inputs.

String

String to replace the PSTRING field on the form. <none> if unused.

Field

Field to replace the PFIELD field on the for.

Form

The Form may be selected from the list box- or alternatively by selecting Field->Value[X] taken from a field in the data base.

Default

A value of one (1) if the existing value of the field is to be displayed as a default, otherwise 0.

Reserved Data Dictionary Settings

The driver in the "target" makes direct use of some fields in the data dictionary. Using these table#/field# settings for other use will have strange results and is not recommended.

Table 1 (Transaction Table)

Fields in this table may be initialized to default values only. The first fields in the transaction table are reserved for (in order)

1ROCNUM

2Track 2

3Track 1

4Track 3

5SPAN

6Expiry Date

7Customer Name

8Protocol Status

9Card Entry Mode

Table 2 (Totals Table)

No reserved settings, however a fixed ten copies are available. Initialization of fields to default values only.

Table 3 (Terminal Table)

This table is the basis of the build of terminal groups, and may be initialized from the Initial Data Table. There is always only one record in the table.

Table 4 (Issuer Table)

One record per issuer, with the current record selected automatically when a card is swiped.

Table 5 (Acquirer Table)

One record per acquirer, with the current record selected automatically when a card is swiped.

Table 6 (Card Table)

Fixed layout, Dictionary specification currently ignored.

Coffind Primitive

ColFind(value,LowField,HighField)

Both LowField and HighField must be in the same table. This table is scanned for a column with the value 'value' between the two fields. The primfive is normally used to locate the CardTable Column for a card. The result variable is set to 0 if no match is found, or 1 if a match is found.

ColSelect Primitive

ColSelect(Column, Table, Reset)

Selects the relevant column of the table indicated.

Column

The column to use. The transaction table has only one column, the totals table has ten. The other tables (issuers, acquirers etc have one column for each record in the corresponding initialization data base

Table

For comparability, 0 (zero) selects the totals table. The tables are as follows

1Transaction

2Totals

3Issuers

4Acquirers

5Card Ranges

Reset

If reset is 1 then all fields in the column are reset.

CommStat Primitive

CommState(port, value, field)

port indicates the port to be tested

value indicates a value to compare and set the status accordingly.

field (optional) indicates a field to store ComsState Value. This function reads the state of the port store the value in field (if specified) and sets the current function result status to true if the value matches "value".

Date Primitive

Date(commdnd, date-field, time-field)

1Read system date into date field and time field

2Set system date from date field and time field

see also

Dates and Times

Dates & Times are special data types both are stored as special numbers.

Date Fields

A Date field is a two byte number, representing a date since 1 Jan. 1940 to 1 Jan. 2110. Subtracting two dates reveals the number of days between the dates, dividing by 7 reveals the day of the week (Monday =O,Tuesday=1 etc).

When moving to or from a text field a date is converted to a text format of DDMMYY. If a format is used this may be converted to DD/MM/YY by using a currency symbol of/in the format. The text format of a date may be either 6 or 8 bytes long—showing the year as 2 or 4 digits.

Date Fields only contain valid dates. Since every date is stored as a day number, the storing the string 32/01/1980 will give is the actual date 01/01/1980. If data is entered directly into date fields, then dates are corrected in this manner automatically. If you wish to check the was entered correctly, then enter the data to a text field, then move the value to the date and check it is equal to the string.

e.g

```
Repeat
  Print(GetDdte,O)
  ActualDate:=TextDate
  Until ActualDate==TextDate
```

The above example will continue to ask for a date until a valid date is entered.

Time Fields

Time fields may be either two or three bytes representing either the time of day to 2 seconds (two bytes) or 1/100 of a second (three bytes) accuracy.

Moving a time to a 2 byte integer gives the number of two second periods elapsed this day. Moving to a 1 byte integer extracts the 1/100s second fraction (up to 199) Moving to a value or larger integer extracts the total number of tics (1/100s sec) which have occurred prior to the time.

Moving a time to of from a text field results in either HHMIVISSFF when moving to a 8 or more byte field and HHMMSS when moving to/from a six byte field.

This text value may be formatted with a format to give HH:MM:SS.FF

Moving values between data and time fields and other numeric types results occurs without conversion. Moving to and from text values results in conversion. See specific entries for conversion details

Dial Primitive

Dial (phone numberphone number)

The numbers specified must be fields. Immediately following each number field in the data dictionary must be a timeout field then a retry field and then a mode field. The upstream prot is implied.

Do Primitive

Do (Function)

also known as

DoFunc(Function)

This primitive is used to activate another script function as a subroutine call

Event Primitive

Event(Function,system event)

Sets the specified function to be activated whenever the event occurs

Exit Primitive

Exit(Now?, Value)

This primitive is used to set the return value of the current function, and optionally, exit immediately.

The 'Value' is stored in the Calculation Result, which will be regarded by any calling function as a result.

If "Now" is true (is 1) exit will be immediate, otherwise the exit value will be established.

Func Number Primitive

Function#field1number . bad number function)

Execute the function with the assigned function#. Typically this primitive will be used by a user function set to be activated by a Hot Key on the target device labeled "Fn" or "Function" or similar. Such a user function would prompt for a number and then call this primitive (Function#) with that number as a parameter.

See Function Activation.

The "Bad—Number—Function" is a function in the script to be executed if the no function matching the first parameter exists.

This is used to implement number functions—for example clearing memory might be function 1055. The user presses the "Function" hot key, then enters 1055 to execute the function.

To achieve this

1a function containing this primitive should created and set up under function activation to have the appropriate key code.

2A function containing the appropriate action for the numbered function should be created and set up under function activation to have the appropriate function number

The function number also returns the logical result of the request to call the numbered function. i.e false if no function exists, otherwise true.

If Else End Primitives

If

The next primitive is executed. If true then all primitives between the if and the else are executed. If false all primitives between the Else and End are executed. If nothing is required between for false then Else may be omitted.

If ! (if with <not?> parameter)

Else

Optional in an If see above.

End

Marks the end of an If or While. See While End

KeyBd Primitive

KeyBd(mode)—(O=off/I=on)

Upon entering idle state, the action of the keyboard is determined by the last KeyBd command. The keyboard (except cancel) will be ignored if off was specified.

MAC Primitive

mac(key,mode,message,field)

All targets must support the storing and use of 4 64 bit keys.

mode I

Calculates a mac of the 'message'and stores the value in the 'field' specified. Uses the 'key' specified. If the 'message' is 8 bytes in length only (or less) then a single DES enscription only results.

mode 2

Stores the specified key into secure memory from 'field'.

Mod

Mod(Value,Divisor)

The value "value" id divided by the divisor, and the remainder is the result.

e.g.—the following example would set the Data Field "Remainder" to 4. (25 divided by 7 has a remainder of 4.

Mod(25, 7)

=> Remainder

Pin Primitive

pin (field)

Retrieves pin block from pinpad. Not supported on all cardscript devices

Print(Display/Report, Part)

Form

The Display/Report may be selected from the list box—or alternatively by selecting Field->Value[X] taken from a field in the data base.

Part

Values—0=all, 1 pre print/header, 2=body, 3=post print/footer

see Forms—end of Header/PrePrintt & Start of Footer/Post Print

Action

The selected section (or all) of the display is displayed or, in the case of a report, printed.

With displays, any input fields will be accepted, however the Show Primitive is recommended for input operations

ProcDown Primitive

Procdown(protocol, port)

The specified protocol is started on the port specified. This function is normally used for downstream protocols such as

an ECR. This function is intended for more advanced users and the protocol must specify its own success and fail functions.

Protocol Primitive

Prot (protocol, Function 1, Function 2)

The specified protocol is started on the bank coms (up-stream) port. The current function execution continues. KeyBd(Off) is set (it may be overridden). If the protocol returns a value of zero, Function 1 will execute, if any other value is return Function 2 will execute. (A KeyBd(on) will automatically happen before either function.

Range Primitive

Range(field,Min,Max)

Returns true if the value specified is >=min and <=max value

Repeat/Until Primitives

Repeat

Repeat sets the execution point for a following until

Until(Case) Cases are 0=False, 1=True

Executes the next primitive and if the result agrees with Case then the Repeats everything after the repeat primitive.

Report (Form, Function)

Action

First prints any pre print or header from "Form". Then for each transaction in the batch calls "Function" and prints the details section of "Form". After cycling throughout the batch, then prints any post print data from "Form".

Form

The Display/Report may be selected from the list box- or alternatively by selecting Field->Value[X] taken from a field in the data base.

Function

A function to be executed before each detail section is printed. For any transaction in the batch for which the function returns FALSE will be skipped.

Restore Primitive

Restore(layout,field,secondary field)

This primitive is used to retrieve information from the Batch Area.

Layout

This optional ("none" is permitted) parameter specifies which transaction layouts are considered for retrieval.

WARNING! All records searched using a field other than RECNUM are actually retrieved, changing the contents of the data fields in their layout. Using a value of "none" may have side effects!

Field

The primary search field. Searching will advance throughout the Batch Area until a match is found.

Secondary Field

an optional (Use RECNUM for "none") secondary search field.

Rom Function Primitive

Rom(valuelfield, Message)

Generally the parameter passed is passed directly though to the bios. The following values are assigned for portability % The message parameter is ignored unless otherwise stated.

1Go to ROM mode.

2Erase memory and return to Rom

3Start TIVIS download (from ROM mode).

4Store Rom Params.

Uses Message and returns Success status.

See ROM SETTINGS.

51—oad Rom Params.

Uses Message and returns Success status.

See ROM SETTINGS.

6Activate Rom Edit.

Returns Success status. See ROM SETTINGS

see Also

ROM SETTINGS

What are ROM SETTINGS

Sub Heading

Normally target devices store programs in RAM memory, and are capable of loading these programs over the telephone Network. In order to achieve remote loading the device must store telephone number and other details required. The device must have a method of loading and/or editing these details.

Methods vary from device to device with information normally being obtained from the keyboard, a smart or magnetic card or some combination. Obviously the information must be able to be set prior to the application loading.

Communication Between Rom & CardScript

Two possible reasons for CardScript to interact with the ROM Settings arise.

1 The parameters may need to changed in a device already loaded with the CardScript application.

2A CardScript application may need access to the ROM Setting values.

Edit Rom Settings—Rom Function 6.

The desired method of allowing change to the settings is to use this function primitive call.(see Rom Function Primitive.) This primitive may not be supported by all Drivers and (check with the driver provider) but provides the only device independent method of implementing the function. An advantage of this function is the operator sees the same interface as when configuring the terminal prior to loading CardScript.

Load Rom Settings—Rom Function 4.

This function is used to obtain the ROM settings in a Script.

Store Rom Settings—Rom Function 5.

A Script Program may load the Rom Settings with Function 4, allow editing of values and use this function to store the settings. It is recommended to use function 6 (edit) in place of this procedure where available as this mechanism allow changing of device specific settings.

The Rom Communications Buffer.

To provide a much device independence as possible using functions 4 and 5 CardScript defines a standard Communications Buffer Layout with a private area at the end. All Fields are ASCII.

The first three fields are assumed to be used for all communications. 2 bytes connection mode. Lan Leased Line etc 4 bytes station/Lan Address 8 bytes telephone prefix—eg "9," Field 1 16 Bytes Terminal ID. The ID as seen by the software management system and not

necessarily other systems.

8 bytes terminal type

24 bytes phone number

24 bytes connection string

Save Primitive

Save(transaction layout)

Saves the current transaction to the batch using the layout specified. A new Transaction Index is generated according to the method specified by the last TxnIdx: Primitive, with the new index stored in field(0,0) RECNUM. For details on RECNUM see Reserved Data Dictionary Settings.

The number of transactions (of the selected layout) which can be stored is returned. If zero is returned, then the save could not work! If 1 (one) is returned then no more may be saved. If two is returned then only one more may be saved, etc.

Store Primitive

Store(offset,messageLayout)

The store primitive stores the last received message, starting at byte <offset>, using the specified message layout. The function result status is set by the operation. (Set to FALSE if the store did not match).

Tots Primitive

Tots(valuer Field)

Selects the relevant totals column.

This primitive has been replaced by the ColSelect primitive. Old programs are automatically upgraded, since parameter 2 or LineTable, when zero, will select the totals table

TxnIdx Primitive. Set Transaction Index.

Txnldx (Field1,Field2,mode)

Field1 is optional. If include the first two digits of the Index are set from this field.

Field2 specifies the main field on which the Index is based. By default this is the ROC field.

the mode specifies how cardscript increments the Txn Idx.

0=add 1

1=Amex Style

2=None. Incremented by script.

This function would normally only ever be used in a start up function. The calculated value is always stored in the ROC field.

User Function Primitive

The user function primitive is used to call any of a range of functions. The functions call by user function are NOT standard.

Primitive—Extensions

It is possible to extend the primitives available to cardscript. The extensions take to form of a block of 'C' code loaded with the Script. 'C' code, of course, has the restriction of being non portable.

The existence of these extensions is to allow extensions to a set of primitives to be tested without changing the core driver. Any extensions initially tested by this means must be added to the set of primitives in a new release, otherwise the code calling them will never be portable.

Wait Primitive

wait(minutes, 100 msecs)

The current function pauses for then number of minutes+1 Oaths of seconds specified. A delay of up to 1000 minutes (over sixteen hours is possible) and as small as 1/10 of a second.

While/End Primitives

While(Case) Cases are 0 False, 1=True

The next primitive is executed. If the result matches Case then all primitives between the While and the End are executed, then we come back again to the While. If the result does not match Case then execution continues with the primitive following the End.

End

Marks the end of an If or While. See also —.— If End For specific categories of primitives see

Communications Primitives

Data Entry Primitives

Displaying and Printing

For information on configuring CardScript for target device function primitives (advanced users only) see

Configure Function Primitives

For advanced users only!!

Usage—Name Changes

Changing the name of a primitive or a primitive parameter will cause all scripts using the name change to be automatically updated. Both this type of change and any changes to the "infix" status of a function will have no affect on the driver and scripts will function without further change.

Usage—Adding, Deleting, Changing Parameter Types

Parameter Settings

Each function parameter has the following possible categories

1A Field from the data Dictionary

2Numeric Value—Which may be displayed as an index to a file 3A String

Any parameter may legally accept any combination of categories

Layouts

Cardscript allows you to graphically enter your layout specifications. For details on Receipts, Reports, Displays, Messages, Protocols, and Transactions see Layouts Menu

Layouts are the main engine of any application. Although all layouts must bee brought into operation by functions, layouts also in turn launch functions and other layouts.

Form layouts, message layouts, and transaction layouts are similar in operation. All three are an arrangement of fields and strings called a field panel. For details see

Field Panels

All field Panels (Displays/receipts, messages and transactions) have a Panel Control box in common. The selection in the Panel Control box selects the action to take place when the left mouse button is clicked over the panel.

Additional controls are present of some panels, however, this box always contains

An add field—control with field edit box and pallet selector

Clicking on the p'a'nel when [add] is selected adds a new field as displayed in the edit box

Before—clicking on the edit box to set the field to be added, select the appropriate type of item in the "from pallet" drop down list box

Clicking on the edit box brings up either the Select Field Dialog or the Enter String Dialog, in accordance with the pallet selector

A dealt field control

Select [delete] and then click on the appropriate field

A select field Control

Select [delete] and then click on the appropriate field

Field Editing

To edit any field, double click on the field

Forms (Displays and Reports)

see also Field Panel, Print Primitive, Show Primitive and Report Primitive

The Screen is Divided into four sections

The Form (Display/Report) Panel

The panel is a Field Panel where the location of the of each field corresponds to the place actual data will appear on the display/printer

The Dashed Boundary

Depending on the Display/Report type, a dashed line will appear showing the limits of the display or printer. This boundary is drawn in accordance with the settings in the Tables/System menu and can be changed at any time.

Form Name

The display report name is used for reference to this screen and should contain a meaningful name.

Panel Control

In addition to the panel controls discussed in Field Panel, two additional controls are present.—<

<<End of Pre Print

Pre-Print fields appear with a grey background Select this item and click on the field after the end of the header section of the report.

Used in reports, the header section is printed once at the beginning of the report. The sections following the header will be printed once for each transaction in the batch. see Report Function for further details

Used in receipts (see Print Function for further details) used to divide the receipt into sections

Start of post print

Post-Print fields appear with a grey background, and can only be distinguished from Pre-Print fields if there are fields in between. (As would normally be the case.)

Select this item and click on the first field of the post print section of the report.

Used in reports, the Post-Print section is printed once at the end of the report. see Report Function for further details

Display/Report Type

The types are

Display—layout will always appear on the display, and in scribe will have a border reflecting display width and number of lines

Secure Display—reserved for future use

Printout—layout will always appear on the printer, and in scribe will have a border reflecting printer width

Soft Keys

The Soft Keys Button allows selection of a soft key set. see

Messages

Output Messages

A messages buffer is built from fields in the data dictionary, and from strings in much the same way a printout is build. However, in messages, all data may be represented in forms other than ASCII. (see "the message engine". Formats may be used to specify data within the selected representation.

Input Messages

Messages are also used to specify how data is transferred from a received buffer and stored in data fields.

see also

The Message Engine (Processor)

The message engine is used both to transfer information both from data fields into a message buffer, and from a message buffer to data fields

see also

Message Data Mapping

Every field in a message buffer is converted from the type in the data field, to the representation of in the message buffer. For "Forms" all data in the buffer is Ascii, but in other messages the data may be any of the following

Ascii

Ascii Representation

Strings and Text Fields

Strings & text fields are simply copied. If the source is shorter then the destination, spaces are used for padding

Integer

Integer to Asch

For one & two byte integers, the binary value is converted to its string equivalent and then formatted according to any format specified. Larger integer conversion may appear in a later release

Ascii to Integer

Again limited to 1 and two byte integers, the value of the text is calculated and stored in the integer.

Amount

As for integer.

Dates

Dates are converted to either DDMMYY or DDMMYYYY if the Ascii field is longer than 7. Formatting is applied on conversion to ASCII only.

Times

Times are converted to either HHMMSS or HHMMSSFF (where FF is the fractions of a.second in hundredths) if the Ascii field is longer than 7. Formatting is applied on conversion to ASCII only.

Hex

Hex Representation

Amounts

To Hex: The data is converted to a BCD string and then expanded

Integers

The binary value is converted directly to Hex. eg a one byte value set to 35decimal (23 hex) would be converted to two bytes—characters' (Ox32) and '3' (Ox33) representing the hex value 23.

Binary

Binary Representation

Integer

Binary representation of integers is High Byte . . . Low Byte. As a binary value. No Actual conversion takes place

Text

Binary representation of Text is to assume the text is a hexadecimal string and convert this to binary. To get an exact copy of the string use Text Representation.

BCD

BCD Representation.

The Data is converted to the BCD data type.

Formats

Formats are used for specifying exactly how data will be represented

Justification

Any time the data length is less than the field width, the justification will be used to decide where the data is placed.

Allowed Characters

Specifies the type to characters allowed during input.

Minimum Characters

Specifies the minimum characters allowed for entry to a field.

Maximum Characters

Specifies the maximum characters allowed for data entry, and the maximum displayed characters on output.

Input Window

A non-zero value in this field specifies input will occur within a window. e.g A 24 character text field may be input using a 10 character window because of limited display space. Note that only ten characters of the input would then be visible at any one time.

Input Validation

A function may be specified here to valid input using this format. The validation function may store the current input using the —> (res) primitive. See also the Console Primitive command 4.

Note that an input validation function MUST NOT do any displays, as the current display would be overwritten.

Suppress Leading Zeros

Check here to suppress leading zeros in numeric fields, or leading spaces in text fields

Decimal Places

Select the appropriate number of places, and the character to use as the decimal indicator. Selecting the decimal (from '.' or ',') also determines the character for thousands separation. (The opposite character to the decimal is used for thousands.

Check the box to require keying of the decimal indicator during input. If this box is left unchecked, data 1.00 (. as decimal) would be input as 100, cash register style.

Auto OK

If this box is checked, when the maximum characters are entered, input will be concluded.

Thousands

The thousands separator(as determined under decimal places) will be automatically inserted.

Password Mode

The first character of the currency symbol will be be displayed in each position, in place of the actual character entered.

Currency Symbol

Specify if the currency symbol is to be displayed. One or two characters may be entered. If the value is two digits, the digits are legal hex digits then these will specify a the character. e.g. 41 would specify the characters', as would a single A character.

This field as has other uses.

The separator for date & time fields is the first character. For time fields the second character is used to separator the hundredths of seconds when displayed

Length Indicator

In addition to the data, the data length is to be included. The number of digits to use may be 1, 2 or 3. The length may be before (pre) the data or trailing (post). As an alternative to a numeric length specification the currency symbol may be used to indicate the end of a variable length field

e.g.

length as 1 —5abcde is a five character field

length as 2 —05abcde is the same field

currency symbol as ':' and use currency selected

abcde: is the same field again

Pad BCF with F

Check here for BCD fields of odd length to be filled with a trailing F nibble. If unchecked a leading zero nibble would be used.

Transactions

Two other layouts types are also available

Bitmaps

Protocols

Protocols describe message flow both from and to the target device. The top line specifies outgoing messages and the other lines display possible incoming results. A protocol consists of lines and sections.

Request Line

At the start of each section is a line 1 (optional for the first section) which describes the outgoing message. This is the request line.

Response Lines

Lines 2, 3 and above define actions to be taken when a response is received. These are the response lines. A

response may be a data received or a time out. When a timeout occurs the first line with a timeout will be selected, any other line with a timeout will never be used. When data is received, all lines beginning with a message are tested to see if received message matches the requirements.

The first item on each response line must be either a message or a timeout.

Protocol Screen Editing

Adding Entries

Select the desired item to add, then click on the display at the desired location

Splitting Sections

A section may be split on line 1. Click below the line slightly to the left of the field to become part of the second section.

Adding to a Line

When adding fields click on the field that will be after the new field. To add to the end of a line click about 3 spaces beyond the end of the line. Always click on the desired line.

Inserting a line.

Click where the new line should start

Retrys/Skips

After entering a Retry, the retry field will be selected. Or you can select the retry later. Once a retry is selected the <<set retry>> and <<set skip>> commands can be used to set the points where execution should move in the event of either a retry or the retry count being exceeded. Note, retry can only move back and skip can only move forward. For those with color displays, the retry arrow is green and the skip arrow is drawn as red. You can't put a retry/skip on line one.

Identifying Input Messages

The first field of each input line is used to select when the input is appropriate. The following possibilities are catered for

Control

The input line is selected when the a message begins with the specified character

Message

The incoming message is matched against the message specified.

Timeout

A timeout line will automatically be selected in response to an incoming timeout.

Function

If the function returns true the message is matched. The Store Function.

Functions Launched By Protocols

Within protocols it is possible to launch functions for various reasons, particularly to store complex messages and select options.

Such functions should NOT halt operation, either by WAIT(or for input or any other event. Should a function attempt to do so, subsequent functions launched from the protocol, including the "good" and "bad" functions, will execute before the function resumes.

Delay any inputs until the good or bad functions at protocol end. If you are an expert user and must do an input, make it the last command in the function and exercise caution.

Repeated Messages

A protocol may involve repeated messages. That is, after storing the data from an input message, another similar message will be received.

Fixed Number of Repeat Messages

If the number of times a message is to be received is fixed then the following approach may be used

<Msg><Retry(nn)>

Where nn is then number of messages expected

Variable Number of Repeat Messages

If the number of times the message will be repeated will vary, i.e a flag in the message indicates that a repeat message will follow, then the following technique is recommended.

use an input line with <Function><Retry(O)>

This will cause a loop whilst the function returns true. From the function, use the store primitive to save the data and return true if another message is expected

Layout Primitives

Layouts use the following elements as building blocks

Putting it all together

Build Menu

Build Target Group Files

Builds the font and conf files ready for program execution

Build Script as Fragment

Builds a reduced script for loading either onto a smart or through the communications network, for describing a particular operation which may be changed without loading a new program.

Build Secure Prompts

Builds the list of secure displays and associated strings for loading into a secure display.

Run Simulation of Build

Activates the terminal simulator program see also

Files Produced By Build

Font Files

xxx is the number of the font. Currently always zero

fontlxxx.bll

Characters O . . . 127. Bytes are dots across. First byte is top row. If more than 8 dots across, then the next byte continues the dots

fontlxxx.bll

Characters 128.255, using the same format as theTfile

fontdxxx.bll

Characters O . . . 128. Bytes are up and down. First dot is top left (bit 0 of byte 1) then dots down the character.

fontuxxx.bll

Script Fragments

Script fragments are small scripts (usually <256 bytes) built separately to a main program. Theses scripts may then be loaded into the terminal (either from a smart card or as part of a message) in order to specify operation of changeable program feature

EXAMPLES OF FRAGMENTS

A Fragment for User Authentication

A Smart Card could contain a program fragment specifying how the terminal should check the user of the card is the real owner. Then cards may be issued with varied scripts such as

Input & Check PIN

Print A Slip and request a signature Do nothing—no check

A Fragment for Communications Protocol

A server could have a list of communications protocols or various networks. Then the terminal could dial the server and request the relevant fragment for a particular network (either because the terminal has no information on the network—or the existing protocol no longer functions), allowing the terminal to operate on a new or changed network with obtaining a complete new application.

Menus

File Menu

Configure Menu

The configure menu is greyed on standard level Scribe. The functions available are for the use of advanced users only. Generally within an Organization using CardScript either one master user will be placed in charge of setting configurations or configurations will be set by an external consultant.

The configuration options are

Configure Simulation

Host Comms

Select a comm port for the simulator to use for modem communications. If none are available select "none". Selecting "none" precludes testing comms facilities

Terminal Group

The Build process creates several build files one for each terminal group. The setting chosen here determines which build file will be used for simulation.

Cards

The simulator does not use a real card reader. Instead it supplies card data from this table. Enter card data as required for up to eight test cards.

To provide the simulator with information on this PC and to create test "Cards".

Configure Targets

A number of target machines may be described to the cardscript system. The information about the target machines is used in various places throughout the scribe system to present information in a manner appropriate to a currently selected "target" machine. see System Record information for setting a Current target.

Object Types

The target machine is described by arranging an number of "objects" on this panel. Their is one of each of the display, printer, and reader objects, and as many button objects as are appropriate.

The display object is used to specify the display configuration in lines and columns. This object should be dragged to an appropriate location in the window.

The printer object is used to specify the print width columns. This object should be dragged to an appropriate location in the window. The number of lines setting bears no relation to the number of lines on a printer page, this field determines how many lines will be available for viewing during simulation.

The reader object is used to describe which area of the window will be used to display buttons for simulation of a card reader. This area bears no relationship th an actual card reader. Drag this object to an otherwise unused area of the window.

Any number of button objects. The object correspond to the push buttons on the target device. Five different button styles are available. Configure these styles as required. When a style is changed, all buttons of that style will change in appearance. Keycodes returned should match those returned by the actual terminal B10S. Use the KeVMap Primitive to force the map these codes to the codes required by the actual application.

For describing the various hardware platforms to the system

For designing Tables Screen Layouts and Contents used on the PC with Scribe

For designing the Data Dictionary used in the Target Device

For Specifying the functions available within the target device.

REFERENCE

Index

Glossary

#

"->" (goes to) primitive: <goes to primitive>

"KeyBd": <KeyBd Primitive>

"refer":<ColSelect Primitive>

"target": the PC, EFTPOS terminal, PiNpad or cash register which will be used to run the

developed application.

—> (res): <result goes to primitive>

B

Batch Area: Storage area of memory. Used for storing transactions and any other miscellaneous data. Also may be thought of as file storage.

bitnumbering: <Bit Numbering>

C

CardEntryMode: <Card Entry Mode>

ColSelect primitive:

Commands for Memory Cards: <Commands For Memory Cards>

Console Primitive: <Console Primitive>

D

Data Dictionary Field Attributes: <Data Dictionary Field Attributes>

Date & Time Fields: <Dates and Times>

E

eserved data dictionary fields:

F

Field Panel: <Field Panels>

Forms —end of Header/PrePrinft & Start of Footer/Post Print:

<Forms>

Function Action: <Function Action>

Function Actions: <Function Actions.>

Function Primitive Categories: <Function Primitive Categories.>

Function Primitives: <Function Primitives>

H

HEX: . Digits 0.9 and A.F

I

Initial Data Field Attributes.: <Initial Data Field Attributes>

K

KeyMap Primitive: <KeyMap Primitive>

P

PFIELD: special field for use on Forms. In place of this field a supplied parameter field will be displayed

Print Primitive: <Print Primitive>

PSTRING: special field for use on Forms. In place of this field a supplied parameter string will be displayed.

R

RAD: Application Development (especially used with 'tool')The process of defining a program in a very short time by starting the program definition with the user interface.

Report Primitive: <Report Primitive>

Reserved Data Dictionary Settings:

Reserved Data Dictionary Settings: <Reserved Data Dictionary Settings>

ROM SETTINGS: <ROM SETTINGS>

S

Show Primitive: <Show Primitive>

Smart Card Type Code: <Smart Card Type Codes>

System Table Settings: <System Table Settings>

T

Txnldx Primitive: <Txnldx Primitive>

W

Windows: popular operating system for PCs. based on an event driven architecture.

APPENDIX B

Bios Objectives

The objective of the Cardsoft BIOS is to make all devices used for running Point of Sale software compatible with CardScript programs.

Bios Usage

The Cardsoft BIOS specification is designed to allow the creation of portable programs for Payment Terminals. Any given implementation of the BIOS will encompass its own "look and feel" which, in turn, is imparted to applications using the system. This is possible since the BIOS specifies what must be achieved by low level functions, rather that the manner of achievement, This means that not all implementations of the BIOS are equivalent and there is scope for vastly different performance and operational convenience whilst still maintaining BIOS compatibility. As an example, the BIOS itself does not specify how such things as how cursors and editing functions are implemented, there is simply a call specifying display this field and allow it to be edited. Thus the field editing rules are determined by the individual BIOS implementation. One brand of equipment over type may be standard, on another insert may be the default.

This leaves individual implementers with the ability for creativity and a framework which allows for the performance and convenience of their programmers to be a product advantage.

Also supplied in addition to the core BIOS are some implementation routines. These are supplied in source code as a starting point for actual implementation. However the code in these routines is not applicable to all hardware configurations and would expect over time to be modified in any given implementation.

The BIOS described in this manual represent the interface between Cardsoft EFT applications (including the driver for CardScript) and an EFT terminal, however the specification is general purpose in nature and may in future be used to support other systems. This manual assumes CardScript is to be supported and is geared to assist in achieving this goal. In addition to the functionality described here the EFT device must have its own "bootstrap" system. Where Cardsoft applications are being added to existing products a software module which interfaces between routines described here and the existing driver software can easily be produced. This BIOS specification remains the property of Cardsoft.

Utilising Existing Operating Systems

When first adding CardScript to a device, some level of BIOS or operating system will normally be already in place. In many cases it is desirable to add CardScript to devices originally developed years prior with well tested hardware device drivers. In these instances the BIOS will constitute an interface between the existing operating system and the CardScript driver. The BIOS may then be linked with the Driver and the combined application loaded as one conventional application.

The BIOS is designed to be able to be placed as a layer above any pre-existing operating system, and be loaded together with the Driver program an one application to devices installed in the field.

New Products

In the case of new products, created for use with CardScript, a purpose build BIOS will minimize memory requirements and speed time to market.

Steps To Implementation

The steps in implementing the Cardsoft BIOS are as follows. Check the BIOS library supplied with this manual is correct for your microprocessor development tools. Other versions of this library for other development environments may be obtained from Cardsoft.

Choose appropriate optional code. In order to simplify implementation of the BIOS sample code is supplied for some typical hardware configurations types providing

higher level functionality and simplifying installation. It is recommended to make use of this software initially, and replace code as desired once the system is operation on the target hardware. Use supplied outline "main.c" and compile & link.

Add routines to eliminate unresolved externals. Use empty routines supplied for routines to be supplied later. It is recommended to initially include real keyboard and display routines and then add others.

Concepts

Event Driven Structure

Applications constructed to run on the Cardsoft BIOS must be event driven. This allows the BIOS or operating system to have control during idle periods. When an input event occurs, the application is called to process the input, and then returns to the BIOS/operating system. The application sets which routine will handle each message and what messages are enabled.

This event driven structure allows the BIOS to operate as an interface layer to event driven operating systems without problems. Where the underlying structure is not event driven This enables the BIOS functionality to be matched by either low level BIOS code or by a high level operating systems ensuring maximum portability of Cardsoft application, and enabling sophisticated underlying structures to be utilized where present. The event driven structure of the system means that applications do not contain a "main" procedure. Applications have an init-application(routine which sets up a table of routines addresses to be called in the case of external events occurring.

Callbacks

Callback Control—Input

The BIOS must maintain a callback address table with four entries for each input/output file. Associated with each entry in the table is an enable status. When the corresponding event occurs a callback should be made using the address from the table. Each callback contains an optional Code and Message (see below). The BIOS should not issue a second callback while another callback is in progress. This can lead to race conditions.

Output

For porting the Cardsoft Bios to a new machine see

Low Level Interface

The low level interface represents the routines that must be custom written when porting CardScript to a new device. These routines assume that the standard Console module, or equivalent are used. Use of these modules eliminates most of the work in implementing the Cardsoft Bios, but postpones fine tuning the Bios to make use of Specific hardware in the most efficient manner.

Standard Modules

The following modules implement the high level interface

console.c

callback.c

math.c

To implement the low level interface, a single module may be created interfacing the foiling routines to the actual hardware or existing drivers. the categories of routine are

Low Level Display

void dispbin(uchar ch)

Display character ch at current cursor position and advance cursor one place

void cleardisplay( )

void dispStr(uchar *str,in len)

continuous dispbin for length of str.

length of str is either len characters, or if len=-1, then

str is null terminated.

uchar dispScroll(uchar direction)

Directions are

1 left

2right

3up

4down

Each call is a request to scroll one place in the specified direction. The result indicates the success of the request (1=OK, 0= can't do)

Low Level Printer

The only printer routine is low level. see

Printer

void prch(uchar ch)

This routine simply prints the character "ch" on the printer device.

Special codes are as follows

O×A (10) end of line

O×C (12) end of form. Feed I lines as required for tear off of receipt

10 General Routines

uchar softKeyBase(uchar select)

By convention returns V for a parameter of 0, and 'a' for a parameter of 1. Change these to indicate actual key values for soft key sets

buz

void buz(in freq, in duration)

Communications

The following routines must have the code inserted to call the low level drivers correctly.

All routines work with a comfile number. Number 0 is the default and is used for the modem. Number I should be the auxiliary com port, if present. Number 2 is the second auxiliary (again if present) etc.

Sendcom_msg

This routine sends block of characters to the specified port. If low level drivers (such as those used with HDLC) require the block at one time then you will need to call those drivers directly from here. If the target device supports only character mode communications, then the "sendcom" routine may be called once for each character.

```
void sendcom-msg(in comfile,uchar *buf,in countOfChars)
{
}
```

Sendcom

A singe character is transferred to the specified port.

```
/* individual cons character send routine */
void sendcom(in comfile,in ch)
{
}
```

Dial

This routine is used to start the dial process. "num1" is to be dialed "cnt1" times, then if this fails, "num2" is to be dialed "cnt2" times. "Mode" indicates the compunction mode to be used. These parameters are under direct control of the application programmer, but by convention

model=async, 2=HDLC

```
/+MODEM */
void dial(uchar *num1,uchar len1,uchar cnt1 ,uchar *num2, uchar len2,uchar cnt2,uchar mode)
{
}
```

Hang-up

The equivalent of the ATH command on a hayes modem.

```
void hangup( )
{
}
```

txstate( )

uchar txstate( )

This return should return a status as follows

0=busy

1Ready

2Reserved for errors, no currently used

Real Time Clock

The real time clock is red & set with the biosDate Time( ) routine

biosDateTime( )

unsigned in biosDateTime(command , buffer)

Command (1=read Datetime, 2=set date & time from buffer

Buffer DDMMYYYYHHMMSSFF

TPDU—The smart card interface

uchar driveTPDU(uchar 11, uchar 12,uchar *Command, uchar *sendbuf,uchar *receiveBuf)

This routine implements, both the Scribe TPDU and SmartCard primitives. To decide which is call is being made, the Command parameter must be tested.

Command ==NULL, SmartCard Primitive

11, and 12 are the parameters. Refer Scribe.hlp for details

Command!=NULL–TPDU primitive

This a direct implementation of the scribe TPDU primitive,. with L1 as the length of the sendbuffer, and L2 the length of the receive buffer.

If L1 is non zero, there is data to send to the card. If L2 is non ZERO, then data from the card is required.

Result

Return zero, unless the function is used as the SmartCard Primitive, and a result is required.

Timer

Cardscript requires the target device to have a 100 millisecond timer. This timer should may a call to the script routine "time-tick( )"

It is recommended not to make a call direct from the hardware timer interrupt. This would result in actions launched by time tick() to execute with interrupts off, giving some very strange results.

Instead set a flag in the interupt handler and have the event loop clear the flag and call time tick( ) (if using an interrupt handler).

The script driver includes the routine "start-bomb( )" which may be called by the bios interface if required

The Font File

The font file consists of sets of entries as follows

1 Character code—1 byte

2Width—1 byte

31-length—1 byte

4 Bitmap

The bitmap is arranged as follows

For each row of height as many bytes as needed for the bits (1 for width <=8, 2 for width <=16 etc).

Left most bit in the MSB of the first byte.

The file is appended with a block of three zero bytes. (Code, width, Height=O) and no bitmap.

For fine tuning an operational Bios see

BIOS Specification (High Level Interface)

By Category Routines are:—

Console (Display & Keyboard)

Input

Sequential (Non event driven)

Since the non event driven machine must be made to appear event driven, the bios interface must include the main line and call the application to handle any events

```
Sample main( )
{
init - hardware( ); /* perform any hardware specific
initialisation */
init - applications( ); /* call to routine in module DRIVE */
for(;;)
{
    if(event) /* test for event*/
    { clear - event( ); /* clear event status */
    handle-event( )*/ call cardscript event handle - see
list*/
}
........
    }
}
```

Events and Handlers

The following list of events should be catered for

Events List

Console

If the high level console is used. Keyboard events are reduced to a single call Process—

Process-key

void process key(uchar key-code)

All that is required by the implementer is to may key codes from the actual machine to the those to be seen by the cardscript application.

Please note that the application programmer has the ability to remap the keys sing cardscript.

Special Key Codes

O×A "Enter" or "OK". (Completion of input)

0×B "Clr" or backspace

0x1B Cancel

The only other console input is the magnetic card reader.

Please use

callback(Console, 3,<unused>,buffer,<unused>);

(use 0 (zero) for unused parameters. or

process-card (<unused>, buffer)

Magnetic Card Read Buffer

The buffer may include any or all of the following sections. They must appear in order.

Section I (optional)

Identifier byte (001)

Track 1 Data—all ascii values

Track2 (optional)

Identifier byte (002)

Track 2 Data—all ascii values

Track 3 (optional)

Identifier byte (N03)

Track 3 Data—all ascii values

End of data marker

Identifier byte (0x0)

System

System Events

For each event, make a call to the routine

void systemEvent(uchar event)

Communications

Comms Events

Character comms

callback(Port,1,Character,NULL,O)

Message based comms

callback(port,2,0,BufferAddress.MessageLenght)

Dial or Tx Finished

When any operation which made the communications port busy has finished, it should tell the script driver by the following call

callback(port,4,0,NULL,0)

see also

Event Driven Input

Please consult Cardsoft for further information

Structural

Memory Management

The MEMPTR type

External Memory

Often target devices have 8 or 16 bit microprocessors which can address limited memory without the use of paging. To allow access to such memory, the concept of External Memory has been defined

It is not assumed that this external memory is directly addressable by the CPU, instead this memory is accessed only via the memory management functions.

The script, the data dictionary fields, any optional fonts and the file storage area are all stored in "external" memory. These areas of external memory are allocated numbers as used in the getbase(function,

A type MEMPTR is used to address this external memory. In the include file custom.h the type MEMPTR must be defined. If has less than 64k of memory allocated amongst the external memory areas MEMPTR could be defined as unsigned integer. If the memory is larger than this than MEMPTR will normally be defined as "long".

Each block of external memory must APPEAR to be continuous. That is incrementing a MEMPTR with the c++operator must always generate a pointer to the next byte of the area.

The memory management routines must map these virtual memory addresses in to real memory addresses

getbase(base)

The function getbase returns the virtual memory address of each of the following blocks of memory

1 The smart card execution buffer

2The Script area—(includes the initialized data dictionary tables

3The Uninitialized data dictionary table area

4The file/batch area

getDataByte

Returns the byte at the virtual address

uchar getDataInt(offset)

Returns the two byte value at the virtual address specified. The format of the two byte value is always Low/High regardless of the byte ordering of the microprocessor.

getScriptData

void getScriptData(uchar *bufferMEMPTR offset,in size)

Transfers data from the buffer to the virtual address "offset"

setScriptData

void setScriptData(uchar *buffer,OFFSETTYPE offset,in size)

Either set external memory to NULL bytes or to a copy of a buffer buffer is the memory buffer in the standard memory area OR if NULL then the operation is like a memset

add notes on offset type

size is the number of bytes to store

For customizing the cardscript command set set

Adding Function Primitives

All function primitives have up to four parameters. Each Parameter is of either one or two bytes length.

Numeric value parameters of values O . . . 127 are one byte in length. Numeric values of greater length and in the general format, with a maximum value of 4999.

All other parameters are in the general format, sixteen bits High order first

bit 15 set—numeric value all other 15 bits contain the value high nibble=0x5 next three nibbles give string number.

all other values high byte=table, low byte=field.

REFERENCE

Index

Glossary

#

"start bomb( )": <start-bomb>

"time-tick( )": in the script-driver for processing 1/10 second time ticks

The invention claimed is:

1. A communications device which is arranged to process messages for communications, comprising a virtual machine means which includes:

a virtual function processor and function processor instructions for controlling operation of the device, and

message instruction means including a set of descriptions of message data;

a virtual message processor, which is arranged to be called by the function processor and which is arranged to carry out the message handling tasks of assembling the messages, disassembling messages and comparing the messages under the direction of the message instruction means that is arranged to provide directions for operation of the virtual message processor, whereby when a message is required to be handled by the communications device the message processor is called to carry out the message handling task,

wherein the virtual machine means is emulatable in different computers having incompatible hardwares or operating systems,

wherein said device is a payment terminal device and wherein said virtual message processor is used to communicate with peripheral units associated with said device.

2. A device in accordance with claim 1 wherein said device is a payment terminal device and wherein said virtual message processor is used to communication with an application associate with said device.

3. A device in accordance with claim 1 wherein said wherein said virtual message processor is used to communicate means associated with the device for implementing cryptographic series.

4. A new device in accordance with claim 1 wherein said device is a personal mobile device.

5. A device in accordance with claim 2 wherein said virtual message processor implements secure communication services.

* * * * *

CERTIFICATE OF SERVICE

The foregoing nonconfidential Opening Brief and Addendum for Defendants-Appellants VeriFone, Inc., Hypercom Corporation, and VeriFone Systems Inc. was electronically filed with the Clerk of the Court for the U.S. Court of Appeals for the Federal Circuit by using the appellate CM/ECF system on February 18, 2014. All participants in the case are registered CM/ECF users and that service will be accomplished by the appellate CM/ECF system.

Dated: February 18, 2014

Respectfully submitted,

Robert W. Kantner
Jones Day
2727 North Harwood Street
Dallas, TX 75201
(214) 220-3939
rwkanter@jonesday.com

By: /s/ E. Joshua Rosenkranz
E. Joshua Rosenkranz
Mark S. Davies
Richard A. Bierschbach
Brian D. Ginsberg
Cam T. Phan
Orrick, Herrington & Sutcliffe LLP
51 West 52nd Street
New York, New York 10019
Telephone: (212) 506-5000
Facsimile: (212) 506-5151
jrosenkranz@orrick.com

Counsel for Defendants-Appellants

CERTIFICATE OF COMPLIANCE UNDER FEDERAL RULES OF APPELLATE PROCEDURE 32(a)(7) AND FEDERAL CIRCUIT RULE 32

Counsel for Defendants-Appellants VeriFone, Inc., Hypercom Corporation, and VeriFone Systems Inc. certifies that the brief contained herein has a proportionally spaced 14-point typeface, and contains 12,714 words, based on the "Word Count" feature of Word 2007, including footnotes and endnotes. Pursuant to Federal Rule of Appellate Procedure 32(a)(7)(B)(iii) and Federal Circuit Rule 32(b), this word count does not include the words contained in the Certificate of Interest, Table of Contents, Table of Authorities, and Statement of Related Cases.

Dated: February 18, 2014

Respectfully submitted,

Robert W. Kantner
Jones Day
2727 North Harwood Street
Dallas, TX 75201
(214) 220-3939
rwkanter@jonesday.com

By: /s/ E. Joshua Rosenkranz
E. Joshua Rosenkranz
Mark S. Davies
Richard A. Bierschbach
Brian D. Ginsberg
Cam T. Phan
Orrick, Herrington & Sutcliffe LLP
51 West 52nd Street
New York, New York 10019
Telephone: (212) 506-5000
Facsimile: (212) 506-5151
jrosenkranz@orrick.com